PoUDR: Proof of Unified Data Retrieval in Decentralized Storage Networks

Zonglun Li*§, Shuhao Zheng*§, Ziyue Xin*, Junliang Luo*, Dun Yuan*, Shang Gao†, Sichao Yang‡, Bin Xiao†, Xue Liu*
*McGill University
Email: {zonglun.li, shuhao.zheng, ziyue.xin, junliang.luo, dun.yuan, xueliu}@mail.mcgill.ca
†The Hong Kong Polytechnic University
Email: {shanggao, csbxiao}@comp.polyu.edu.hk
‡Oortech
Email: ysc@ntlabs.io

***Abstract*—Decentralized storage networks, including IPFS and Filecoin, have created a marketplace where individuals exchange storage space for profit. These networks employ protocols that reliably ensure data storage providers accurately store data without alterations, safeguarding the interests of storage purchasers. However, these protocols lack an effective and equitable payment mechanism for data retrieval, particularly when multiple data queriers are involved. This necessitates a protocol that ensures both data integrity and fair compensation for data providers.**

In decentralized storage, data is fragmented into small blocks and stored across multiple nodes, a process known as data swarming. Due to this property, traditional data exchange protocols are inadequate in terms of communication and economic efficiency.

We propose the Proof of Unified Data Retrieval protocol (PoUDR). PoUDR incorporates ZK-SNARK to facilitate a fair data exchange protocol. PoUDR reduces the number of blockchain transactions for both single block and data swarming retrieval. The protocol requires only a single key-revealing transaction submitted by the provider to the blockchain for each data block. This architecture allows for further optimization of transaction numbers through a batched proof technique on the provider's side. This approach necessitates only N_P transactions within a specific time frame when data consisting of N_D blocks, provided by N_P providers, is queried by N_Q queriers.

This work provides a comprehensive definition for *Secure Swarming Data Exchange (SSDE)*, including security assumptions. Also it offers a detailed game-based security analysis for the PoUDR protocol. Moreover, the PoUDR protocol has been fully integrated into the Bitswap protocol (IPFS). Within this integration, our proposed Relaxed Groth16 algorithm addresses the significant technical challenge of generating zero-knowledge proofs, leading to substantial cost reductions for overall feasibility of secure data retrieval in decentralized storage networks.

I. Introduction

Exchange of data for money online presents the challenge of establishing mechanisms that ensures fair trade between sellers selling data to buyers [28], [10]. Fair refers to equitable transaction outcomes where a seller gets paid upon delivering the specified data and a buyer pays only after receiving the correct data [13]. The challenge arises from the fact that mutual trust ought not be assumed naturally. As noted by Pagnia and Gärtner [34], such fair trades require a trusted third party for mediation, otherwise designing protocols that ensure such fairness is unfeasible. As further research has been conducted on fair data exchange mechanisms employing decentralized blockchain, various protocols have been developed to tackle the aforementioned challenge by using the blockchain to function as a trusted third party [24], [7], [11], [12].

We are examining our work from two perspectives: one focuses on enhancing protocol-level performance and features compared to the aforementioned protocols, and the other addresses practical technical challenges and associated costs to ensure feasibility upon deployment on IPFS, problems that other protocols have not thoroughly addressed.

In the context of IPFS, Files are segmented into smaller blocks and then disseminated across various nodes within the peer-to-peer network. When a user requests a file, these protocols use swarming retrieval mechanism, aggregating the necessary blocks from multiple nodes simultaneously as shown in Figure 1. IPFS distinctly identifying each fragment of the file through its cryptographic hash [2]. The secure swarming data exchange for money problem discussed in our study emerges in a process involving queriers $\mathbb{Q}$ and providers $\mathbb{P}$. Considering a data file D is fragmented into N_D blocks and each provider $\mathcal{P}_i \in \mathbb{P}$ hosts a subset of the blocks, $B_{\mathcal{P}_i}$, with possible overlaps among other providers. One querier $\mathcal{Q} \in \mathbb{Q}$ issues a query q for D, triggering a subset of providers $\mathbb{P}_q \subseteq \mathbb{P}$, to collaborate to assemble and deliver the entire set of N_D blocks constituting D. The blocks provided to q by a provider $\mathcal{P}_i \in \mathbb{P}_q$ is denoted as $B_{q,\mathcal{P}_i}$. This collaboration ensures that $\bigcup_{\mathcal{P}_i \in \mathbb{P}_q} B_{q,\mathcal{P}_i} = D$ for data correctness, and ensures that each provider receives payment for their specific contribution to q.

At the protocol-level, one of the objectives is to optimize this process, reducing the number of blockchain transactions required to function as a trusted third party for achieving fair trade. Another objective is to tackle the problem of unauthorized key sharing. Protocols such as ZKCP are unable to address the unauthorized key sharing (key reselling) issue [30], [21], as outlined in Table I. The issue arises when a decryption key, once sold, remains valid and could be further reused or distributed to others to decrypt the same or other data blocks, and thus avoid payment leading to unfairness, which has been a subject of longstanding scholarly [3]. Circumvents

§These authors contributed equally to this work.

mechanisms designed to prevent key reselling, even though cannot prevent data from reselling since off-line agreements could operate outside the system's control, still have positive impact on the resource utilization of providers. Protocols like OptiSwap [12] can address this issue, will incur escalated costs in attaining fairness under this retrieval process due to the focus on cryptographic schemes for individual files. Since adapting those file-level fair data exchange solutions considering a block as a file would lead to the costs attributable to the markedly increased number of transactions required.

At the deployment-level, we explored the technical challenges and cost considerations involved in deploying these protocols on the IPFS platform. The primary challenges in enabling fair exchange on IPFS involve the integration of zero-knowledge proofs (ZKPs) with IPFS block content identification. The generation of a proof that a specific block corresponds to a given content ID, which is typically highly computationally expensive in ZK, as each block to content ID calculation requires thousands of SHA256 block computation. Therefore, the objective is to develop a mechanism for ZKP integration with IPFS to streamline the content ID verification process and reduce overall computational costs. We outline in below I-A how our protocol meets these objectives, addressing the aforementioned problems from both the protocol enhancement and practical implementation perspectives.

A. Contribution

ADDRESSING KEY RESELLING IN DATA SWARMING. In data swarming retrieval, the encryption key for each data block must not only differ for various blocks but also vary for the same block when queried by different queriers, to prevent key reselling attacks. Prior protocols have not specifically addressed this issue. The PoUDR protocol implements a strategy to combat the key reselling problem in data swarming contexts through a combination of private key encryption and commitment schemes. This protocol ensures the independent creation of encryption keys, each uniquely committed and encrypted, making each data block exchange distinct and eliminating the possibility of key reuse. The implementation of the secure and zero-knowledge (ZK)-friendly One-Time Pad (OTP) cipher over finite field $\mathbb{F}_p$ and the key derivation function `PBKDF2-Poseidon` significantly diminishes the computational burden of generating ZK proofs on the provider side, enhancing the protocol's practicality in real-world decentralized storage networks.

REDUCING ON-CHAIN TRANSACTION OVERHEADS. The PoUDR protocol reduces the quantity of on-chain transactions for both individual block retrieval and data swarming retrieval through its batched key-revealing proof approach. In the case of a single block, the protocol decreases the transaction count to just one. For a specific period in which N_Q queriers request data from N_P providers for a file comprising N_D blocks, the total on-chain transactions correspond to the number of providers N_P. This is contrast to other protocols that necessitate transactions equating to twice the number of blocks transmitted, that is, $2N_DN_Q$, as illustrated in Table I. Given that the number of providers holding file blocks is usually substantially lower than the total number of blocks transmitted, especially in scenarios involving large files, this optimization reduces the number of on-chain transactions. This markedly lowers the gas cost for real-world data swarming retrieval.

DEFINITIONS AND GAME-BASED SECURITY PROOF. A formal definition for *Secure Swarming Data Exchange (SSDE)* and the PoUDR protocol is presented, accompanied by a comprehensive security analysis. This analysis is grounded in various attack games designed to deviate from the protocol, demonstrating that the PoUDR protocol meets the security requirements of *SSDE*. Furthermore, we propose strategies to mitigate Sybil attacks targeting data providers in data swarming retrieval contexts, which involve economic deterrents, such as time-locked staking, to enhance security.

TACKLING COST CHALLENGE FOR FEASIBILITY Existing protocols (in Table I) have not fully addressed the complexities of fair data exchange when employing decentralized storage. When implementing a data exchange protocol built on decentralized storage such as IPFS, substantial technological challenges arise. The challenge arises from the necessity to generate zero-knowledge (ZK) proofs for IPFS blocks to verify content IDs, where each block pointed to its content ID must be validated through a computationally intensive ZK circuit. This proof generation renders the system infeasible without significant cost reductions. To overcome this challenge, our solution leverages recursive zero-knowledge succinct non-interactive arguments of knowledge (recursive zk-SNARKs), which reduce the computational overhead by folding the SHA-256 circuit in a recursive manner. This approach significantly reduces the ZK circuit size and thus greatly amortizes the memory overhead. Furthermore, this folding-based optimization is applicable to all kinds of ZK-unfriendly circuits with repeated modules, such as other commonly used hash functions, non-native elliptic curve computation, etc.

RELAXED GROTH16 WITH FORMAL SECURITY PROOF In current decentralized storage projects, each piece of data is assigned a content ID, typically a ZK-unfriendly hash value of the data. Existing data exchange protocols require proving the hash computation in a zero-knowledge proof, which results in unacceptably high costs for proving a ZK-unfriendly hash value. This paper introduces the Relaxed Groth-16 protocol, based on relaxed R1CS [25], to recursively prove hash computations. Additionally, this work provides a formal security proof for the Relaxed Groth-16 protocol and implements the NOVA folding scheme [25] and Relaxed Groth-16 protocol in the Go language.

CROSS-PLATFORM COMPATABILITY. The PoUDR protocol attains cross-platform decentralized storage compatibility by employing a modular approach, conforming to prevalent modules and data formats in these systems. This focus is particularly on the concepts of data block segmentation and decentralized data retrieval. Complete implementation and integration of the PoUDR protocol within Bitswap have been accomplished, with its functionality tested in the actual IPFS

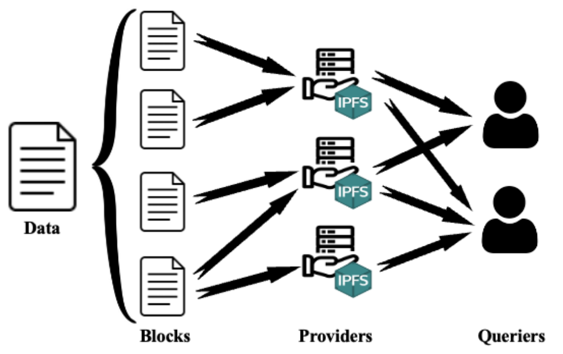

Fig. 1. Overview of Data Storage and Retrieval Pattern in Decentralized Storage Network

TABLE I
Comparison of *PoUDR* and other protocols.

	PoUDR	ZKDET [35]	ZKCP [31]	ZKCPLUS [27]	OptiSwap [12]
#Txns/Data Block	**1**	2	2	2	2+
#Txns/(N_P, N_Q, N_D)-Slot	$\boldsymbol{N_P}$	$2N_DN_Q$	$2N_DN_Q$	$2N_DN_Q$	$2N_DN_Q+$
Key-Reselling Resistance	✓	✓	✗	✗	✓
Data Swarming Friendly	✓	✗	✗	✗	✗
Feasibility-compatible in IPFS	✓	✗	✗	✗	✗

setting to confirm its practicality. Such implementation guarantees straightforward integration into various decentralized storage environments.

B. Related Work

Zero Knowledge Contingent Payment (ZKCP) proposed by Maxwell pioneered fair data exchange using zero knowledge proofs to eliminate centralized intermediary in 2011 [31]. The protocol enabled payments to be released only when specific knowledge is disclosed by the payee, representing that the payee truly possess a specific secret without revealing the secret itself. Secret refers to information or data the payee has and the payor wants to purchase. Subsequently, ZKCP protocol was first used on the bitcoin network in 2016 [32].

Further, enhancements and extensions for ZKCP encompassed. Banasik et al. [1] constructed ZKCP protocol with transactions only meaning no contracts or scripts, specifically adopted for the users are allowed only to post standard transactions. ZKCP revisited [7] presented by Campanelli et al. discussed a fix to some concerns of traditional ZKCP scenarios, where the verifier is responsible for generating the common reference string (CRS) that will be later used for generating and verifying proofs. Should the verifier engage in malicious activities, there exists a potential to contrive a CRS that enables the extraction of information from the proof. In the protocols outlined in [7] the proof maintains a cryptographic condition where the specific secret information used in the proof remains confidential, the secret information cannot be identified or singled out. ZKCPlus proposed by Li et al. [27] later targeted at improving prover efficiency for parallel computations for making it more suitable for more practical real-world hardware applications.

Within fair data exchange research, a blockchain-related research field is the exploration of using smart contracts as a trusted third party, which is necessary for fair exchange [34]. FairSwap proposed by Dziembowski [11] discussed digital goods exchange by introducing a smart contract-based protocol that circumvents the computational intensity of zero-knowledge proofs. Compared to approaches using zero-knowledge proofs, FairSwap's mechanism: proof of misbehavior tackled the cost associated with expensive ZKPs with the limitation of its potential exposure of transactional details in dispute resolutions. Eckey et al. [12] incorporated an interactive dispute resolution sub-protocol to extend the FairSwap so that the proposed extended protocol OptiSwap reduced communication and computational for the scenarios without disputes. Fujitani et al. [15] discussed related problems of automated service fee collection, which could potentially lead to reduced gas costs since it avoids the interactive dispute resolution process that OptiSwap employs.

II. Background

A. Decentralized Storage

In the Web3 ecosystem, decentralized storage represents a considerable shift in data storage and access methodologies, diverging from conventional centralized systems. In contrast to centralized storage systems, which manage data in specific and often exclusive data centers, decentralized storage disperses data throughout a network of nodes, frequently managed by a diverse group of independent operators. This method reinforces data security, strengthens resilience against outages, and fortifies resistance to censorship. Decentralized systems not only store data but often replicate it across multiple nodes, promoting redundancy and ensuring high availability. These systems employ blockchain technology to augment security and facilitate transparent operations, aligned with Web3's foundational principles of decentralization, privacy, and user sovereignty.

Prominent examples of decentralized storage solutions within the Web3 domain include InterPlanetary File System (IPFS)[*] and Filecoin[†]. IPFS implements a peer-to-peer protocol specially crafted for storing and distributing data in a distributed file system. Unlike conventional web protocols, IPFS utilizes content-based data addressing rather than location-based, enhancing robustness and efficiency. Conversely, Filecoin constitutes a decentralized storage network, rewarding participants with tokens for contributing storage capacity. Filecoin seamlessly integrates with IPFS, adding an incentive layer that fosters data retention and distribution. Collectively, these platforms demonstrate the capabilities of decentralized storage, presenting scalable, effective, and secure alternatives to traditional data storage approaches.

[*]https://ipfs.tech/
[†]https://filecoin.io/

B. BitSwap

As a vital component of the IPFS, Bitswap serves as the protocol for data exchange, facilitating the exchange of data blocks among IPFS nodes [9].

At its foundation, Bitswap operates on a straightforward principle: a node first checks locally for a specific data piece before seeking it externally. If absent locally, the node initiates a request to its peers for that specific data. Peers possessing the data respond with it, or they may relay the request further within their network. The distinctive feature of Bitswap lies in its exchange mechanism. Nodes actively monitor peers' interest in their data, leveraging this information to preferentially send data to previously cooperative peers, thereby creating a data exchange strategy. This strategy motivates nodes to actively share data, fostering a robust and efficient distribution network. Furthermore, Bitswap's algorithm is designed for robustness and efficiency. It minimizes redundant data transfers and adeptly identifies the most direct path to the required data, a crucial feature for reliable data distribution in extensive networks.

C. ZKCP

In the progression of modern decentralized financial systems, Zero-Knowledge Contingent Payment (ZKCP) protocol represents a significant advancement. It offers a robust mechanism for secure and private transactions between different parties. It leverages zero-knowledge proofs to enable the seller $\mathcal{S}$ and the buyer $\mathcal{B}$ to complete the transparent transaction without revealing sensitive details of themselves. Traditionally, this challenge would require a trusted third-party intermediary with sacrifice on security and privacy. However, ZKCP utilizes a smart contract $\mathcal{C}$ implemented on a blockchain network with cryptographic techniques to solve the challenge with an inspiring solution.

1) The process begins with $\mathcal{S}$ generating a zero-knowledge proof that unequivocally demonstrates their possession of the requisite information. This proof is then sent to $\mathcal{B}$, who verifies its validity without gaining access to the actual data. This step is crucial as it ensures that $\mathcal{B}$'s payment commitment is based on the certainty of the seller's knowledge, thereby eliminating the risk of fraud or misrepresentation.
2) Once the proof is validated, $\mathcal{B}$ proceeds to send the payment to $\mathcal{C}$ deployed on the blockchain. $\mathcal{C}$ acts as an impartial intermediary, holding the payment in escrow until the transaction's conditions are satisfied.
3) Upon receiving confirmation of the payment, $\mathcal{S}$ releases the information, which $\mathcal{B}$ can then access and utilize. $\mathcal{C}$ then finalizes the transaction by releasing the payment to the seller. This meticulously designed sequence of events ensures that both parties' interests are safeguarded— $\mathcal{B}$ receives the promised information, and $\mathcal{S}$ is compensated for their knowledge.

The implications of ZKCP are far-reaching, offering a solution to the perennial problem of trust in transactions involving sensitive information or digital goods. In scenarios where traditional mechanisms require intermediaries or rely heavily on mutual trust, ZKCP provides a decentralized alternative that guarantees fairness and security. Its applications extend to various domains, including secure information sharing, digital content distribution, and any transaction where the buyer needs assurance of receiving the correct information upon payment [7]. Related works including [1], [7], [14], [27] have implemented different aspects of experiments on basic ZKCP protocol on its shortcomings, introducing new optimization or modification to the protocol details. In this paper, our motivation also starts with addressing the issue and deficiency in ZKCP protocol to further extend the protocol to better serve as a data exchange technique in the cross-platform decentralized storage aspect.

D. ZK-SNARK

Zero-Knowledge Proofs (ZKPs) [17], [18] are a cryptographic method by which a prover can prove to a verifier that she knows a statement x, without conveying any information apart from the fact that she knows x. Formally, a ZKP consists of three algorithms (P, V, S), where P is the prover, V is the verifier, and S is the simulator. A ZKP has the following properties:

- **Completeness:** If the statement is true, the honest verifier (that is, one following the protocol properly) will be convinced of this fact by an honest prover. Formally, for any x, w such that $x \in L_w$ and any verifier strategy V^*,

$$\Pr[(P(x, w), V^*(x)) = 1] = 1.$$

- **Soundness:** If the statement is false, no cheating prover can convince the honest verifier that it is true, except with some small probability. Formally, for any $x \notin L_w$, any prover strategy P^*, and any w',

$$\Pr[(P^*(x, w'), V(x)) = 1] \leq negl(\lambda),$$

where $negl(\lambda)$ is a negligible function.
- **Zero-knowledge:** If the statement is true, no verifier learns anything other than this fact. This is formalized by showing that every verifier has some simulator that, given only the statement to be proved (and no access to the prover), can produce a transcript that "looks like" an interaction between the honest prover and the verifier in question. Formally, for any x, w such that $x \in L_w$, any verifier strategy V^*, and any w',

$$\{(P(x, w), V^*(x))\} \approx \{(S(x, w'), V^*(x))\},$$

where the approximation symbol $\approx$ denotes computational indistinguishability.

Zero Knowledge Proof of Knowledge (ZKPoK) extends the concept of a ZKP by adding a guarantee that the prover possesses certain knowledge. In addition to the properties of completeness, soundness, and zero-knowledge, a ZKPoK includes:

- **Knowledge Soundness**: The proof system has knowledge soundness if a witness can be extracted from any successful prover strategy. Specifically, there exists a polynomial-time extractor $\mathcal{X}$ such that for all adversaries $\mathcal{A}$, if $\mathcal{A}$ can produce a valid proof with non-negligible probability, then $\mathcal{X}$ can extract a witness w such that $x \in L_w$. Formally,

$$\Pr\left[\begin{array}{c}(R,z) \leftarrow \mathcal{R}(1^\lambda); (\sigma,\tau) \leftarrow \text{Setup}(R);\\ (\phi,\Pi) \leftarrow \mathcal{A}(R,z); w \leftarrow \mathcal{X}(R,\phi,\Pi):\\ \Pi \in \mathbb{F}^{m\times k} \wedge \text{Vfy}(R,\sigma,\phi,\Pi_\sigma)=0 \wedge\\ x \notin L_w\end{array}\right] \leq negl(\lambda).$$

Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (ZK-SNARKs) [20], [6], [36], [16], [37] are a special form of ZKPs that have the properties of being non-interactive and succinct. Non-interactivity means that the proof consists of a single message from the prover to the verifier. Succinctness means that the size of the proof is relatively small compared to the statement, and verification of the proof is fast.

E. Relaxed R1CS

Rank-1 Constraint System (R1CS) are a fundamental component in various cryptographic protocols, particularly those involving zero-knowledge proofs. Traditional R1CS is a system of equations designed to represent computations efficiently verifiable. However, for applications like folding schemes used in incrementally verifiable computations (IVC), traditional R1CS can be overly restrictive. To address this, we introduce the concept of Relaxed-R1CS (Relaxed Rank-1 Constraint Systems), which modifies the traditional framework to better suit the needs of folding schemes, thus offering a more adaptable framework that retains necessary expressiveness for complex computations.

A Relaxed-R1CS is defined over a finite field $\mathbb{F}$. Given matrices $A, B, C \in \mathbb{F}^{m\times m}$, which are typically sparse and contain at most $n = \Omega(m)$ non-zero entries per row or column, the Relaxed-R1CS framework accommodates an instance consisting of a vector $\mathbf{x} \in \mathbb{F}^\ell$ (public inputs), augmented with an error vector $\mathbf{E} \in \mathbb{F}^m$ and a scalar $u \in \mathbb{F}$. The witness $\mathbf{w} \in \mathbb{F}^{m-\ell-1}$ satisfies the system if the following relaxed constraint is met:

$$(A \cdot \mathbf{z}) \circ (B \cdot \mathbf{z}) = u \cdot (C \cdot \mathbf{z}) + \mathbf{E}$$

where $\mathbf{z} = (\mathbf{w}, \mathbf{x}, 1)$ and "$\circ$" denotes the Hadamard product (element-wise multiplication). This formulation introduces flexibility through the error vector $\mathbf{E}$ and the scaling factor u, which are not present in traditional R1CS. This flexibility is crucial for accommodating the combination and reduction of constraint systems in a folding scheme.

III. PoUDR Protocol

In this section, we first provide the formal definition of the system model and argue about the security assumption. Then, we give an overview of the PoUDR Protocol and provide details of its important components.

A. System Model and Security Assumptions

This section provides a formal definition of the **Secure Swarming Data Exchange (SSDE)**.

Let $[n]$ denote the set of all the integers from 1 to n, i.e., $[n] = \{1, 2, ..., n\}$. The security parameter is represented by λ. In a decentralized storage context, for simplicity, we define a file as D, partitioned into N_D blocks of equal size, denoted as $D := \{d_i \mid i \in [N_D]\}$. There are N_P providers involved in storing D, represented by $\mathbb{P} := \{\mathcal{P}_i \mid i \in [N_P]\}$. All blocks stored by provider $\mathcal{P}_i$ are denoted by $B_{\mathcal{P}_i}$, which means if $\mathcal{P}_i$ participants providing D, then $B_{\mathcal{P}_i} \cap D \neq \emptyset$. It is possible for providers in $\mathbb{P}$ to have overlapping blocks. Thus, a query q queried by querier $\mathcal{Q}$ on D requires a subset of providers $\mathbb{P}_q \subseteq \mathbb{P}, s.t.\ D \subseteq \bigcup B_{\mathbb{P}_q}$ to supply all blocks in D. Additionally, the blocks provided to $\mathcal{Q}$ by each provider $\mathcal{P}_i \in \mathbb{P}_q$ are denoted as $B_{q,\mathcal{P}_i}$, ensuring that $\bigcup_{\mathcal{P}_i \in \mathbb{P}_q} B_{q,\mathcal{P}_i} = D$ and $\forall \mathcal{P}_i, \mathcal{P}_j \in \mathbb{P}_q$ where $i \neq j, B_{q,\mathcal{P}_i} \cap B_{q,\mathcal{P}_j} = \emptyset$. For simplicity, we assume a universal price for each data block, meaning each successful data query results in the corresponding data provider receiving t tokens. The goal of the protocol is to make the querier $\mathcal{Q}$ receive correct data D, and all providers in $\mathbb{P}_q$ get the corresponding payments. Below is the formal definition of a secure swarming data exchange protocol.

Definition 1 (**Secure Swarming Data Exchange (SSDE)**). *Formally, given the system model described above, an **Secure Swarming Data Exchange (SSDE)** protocol should satisfy the following two security properties:*

- ***Data Correctness:*** *When a querier $\mathcal{Q}$ query D with content ID cid, she finally receives data D' and the probability that she accepts wrong data satisfies:*

$$Pr\{D' \neq D \wedge \mathcal{Q} \text{ accepts } D'\} \leq negl(\lambda). \quad (1)$$

- ***Guaranteed Payment:*** *During Query q querying on D with content ID cid, the querier $\mathcal{Q}$ received D'. Denote the payment sent from the querier $\mathcal{Q}$ to provider $\mathcal{P}_i$ as t_i. If D only consists of one data block sent by $\mathcal{P}_i$, then*

$$Pr\{D' = D \wedge t_i < t\} \leq negl(\lambda). \quad (2)$$

If the data D consists of multiple data blocks sent by a set of providers $\mathbb{P}_q$, then

$$Pr\{D' = D \wedge (\vee_{i,\mathcal{P}_i \in \mathbb{P}_q}\ t_i < t \cdot |B_{q,\mathcal{P}_i}|)\} \leq negl(\lambda). \quad (3)$$

PoUDR operates under the same security assumptions as the IPFS system. This implies that both data providers and queriers may potentially exhibit malicious behavior. In essence, either party is capable of arbitrarily deviating from the established protocol. Regarding the underlying blockchain infrastructure, it is assumed that the blockchain system itself is secure. This means that all transactions submitted to the blockchain system will be executed through smart contracts, ensuring both correctness and liveness.

Algorithm 1 Rehashing Contract

```
Registry:
  Require ZK-friendly and unfriendly hash values: $cid', cid$. Rehashing proof $\pi_{rehash}$ and rehashing proof verifier $V_{rehash}$.
    if $V_{rehash}$.Verify$(\pi_{rehash}, cid, cid') = 1$ then
        $mapping[cid'] = cid$
    else
        $Abort$
CheckHash:
    Require ZK-friendly and unfriendly hash value $cid', cid$.
    if $mapping[cid'] == cid$ then
        return True
    else
        return False
```

Algorithm 2 Query Protocol

```
Require: Content ID $cid$, Querier's public key $pk^{\mathcal{Q}}$, Encryption proof verifer $V_{enc}$, Signature scheme $Sig$, Private key and public key encryption scheme $\mathcal{E}_{data}$ and $\mathcal{E}_{key}$
1: $\sigma_{query} \leftarrow Sig.\textbf{Sign}(sk^{\mathcal{Q}}, cid)$
2: $D^{enc}, \pi_{data}, pk^{\mathcal{P}}, s^{com}, cid' \leftarrow \textbf{DHT}(cid, \sigma_{query}, pk^{\mathcal{Q}})$
3: if $V_{enc}.\textbf{Verify}(\pi, cid, D^{enc}, pk^{\mathcal{P}}, pk^{\mathcal{Q}}, s^{com}) = 1 \wedge RPC.\textbf{CheckHash}(cid, cid') == True$ then
4:     $msg \leftarrow$ "$Recevied||n||pk^{\mathcal{Q}}||pk^{\mathcal{P}}||s^{com}$"
5:     $\sigma_{msg} \leftarrow Sig.\textbf{Sign}(sk^{\mathcal{Q}}, msg)$
6:     Send$(msg, \sigma_{msg})$
7:     Upon listened event $e$ on chain:
8:         $s \leftarrow \mathcal{E}_{key}.\textbf{Dec}(sk^{\mathcal{Q}}, e.s^{enc})$
9:         return $\mathcal{E}_{data}.\textbf{Dec}(s, D^{enc})$
```

Algorithm 3 Response Protocol

```
Require: Secret & public key pair $sk^{\mathcal{P}}, pk^{\mathcal{P}}$. Provers for encryption and key revealing $P_{enc}, P_{key}$. Commitment scheme $Commit$. Private key encryption scheme $\mathcal{E}_{data}$, Public key encryption scheme $\mathcal{E}_{key}$
1: Upon Receiving Query $(cid, \sigma_{query}, pk^{\mathcal{Q}})$:
2:     if $Sig.\textbf{Verify}(cid, \sigma_{query}, pk^{\mathcal{Q}}) = 1$ then
3:         $s \leftarrow Rand(\lambda)$
4:         $s^{com} \leftarrow Commit(s)$
5:         $D \leftarrow Retrieve(cid)$
6:         $cid' \leftarrow Hash_{zk}(D)$
7:         $D^{enc} \leftarrow \mathcal{E}_{data}.\textbf{Enc}(s, D)$
8:         $\pi_{data} \leftarrow P_{enc}.\textbf{Prove}(s, s^{com}, D, D^{enc}, cid')$
9:         Send$(D^{enc}, \pi_{data}, pk^{\mathcal{P}}, s^{com}, cid')$
10: Upon Receiving Message $(msg, \sigma_{msg})$:
11:     if $Sig.\textbf{Verify}(pk^{\mathcal{Q}}, msg, \sigma) == 1$ then
12:         $s^{enc} \leftarrow \mathcal{E}_{key}.\textbf{Enc}(pk^{\mathcal{Q}}, s)$
13:         $\pi_{key} \leftarrow P_{key}.\textbf{Prove}(s^{com}, s, s^{enc}, pk^{\mathcal{Q}})$
14:         $txn \leftarrow WrapTxn(msg, \sigma_{msg}, pk^{\mathcal{P}}, pk^{\mathcal{Q}}, s^{enc}, \pi_{key})$
15:         SendTransaction$(txn)$
```

Algorithm 4 Verifier Contract

```
Require: $msg, \sigma_{msg}, pk^{\mathcal{Q}}, s^{enc}, s^{com}, \pi_{key}$
1: $Assert\ txn.sender == msg.pk^{\mathcal{P}} \wedge pk^{\mathcal{Q}} == msg.pk^{\mathcal{Q}}$
2: $Assert\ msg.s^{com} == s^{com} \wedge \textbf{Unused}(msg.n)$
3: $Assert\ Sig.\textbf{Verify}(msg, \sigma_{msg}, pk^{\mathcal{Q}}) == 1$
4: if $\textbf{V}_{key}(\pi_{key}, pk^{\mathcal{Q}}, s^{enc}, s^{com}) == 1$ then
5:     $addr^{\mathcal{P}} \leftarrow \texttt{addr}(msg.pk^{\mathcal{P}})$
6:     $addr^{\mathcal{Q}} \leftarrow \texttt{addr}(msg.pk^{\mathcal{Q}})$
7:     Transfer$(addr^{\mathcal{Q}}, addr^{\mathcal{P}}, t)$
8:     $Emit\ event\ \textbf{e}(addr^{\mathcal{Q}}, addr^{\mathcal{P}}, msg.n, s^{enc})$
9: else $Revert$
```

B. Overview of PoUDR

Current Data Retrieval solutions [7], [27] assume the use of ZK-friendly hashing algorithms to ensure data integrity, or that the data size remains small (several bytes). However, most existing decentralized storage projects employ ZK-unfriendly hashes, with each data block significantly exceeding 1KB; for example, IPFS uses SHA256 for computing a data file's content ID and recommends a block size of 256KB [29]. Previous solutions are impractical for ZK-unfriendly hash scenarios since proving the hashing process for large files directly demands prohibitive computational resources (Hundreds of GB for RAM). PoUDR employs the Relaxed-QAP and Relaxed Groth16, which are based on relaxed R1CS [25], to prove the hash process in a folding manner, thereby reducing machine requirements. Furthermore, the Rehashing Protocol, which maps a ZK-friendly hash value to an unfriendly hash value for the same data, reduces proof generation costs by enabling the prover to directly prove the ZK-friendly hash computation.

A trusted third party, such as a blockchain, is required to ensure security and fairness in the exchange of data between two parties [34]. Yet, interacting with a blockchain frequently incurs inescapable time and financial cost. Especially in data swarming scenarios, where a file gets split and scattered among several data providers, numerous blockchain transactions are necessary to retrieve a single file. Consequently, the expense of transferring a single file becomes excessively high. Considering these difficulties, the PoUDR protocol also aims to reduce blockchain interactions and lower the costs related to data transmission in scenarios of data exchange between queriers and providers. The protocol achieves this while maintaining the security and fairness for all parties involved.

Both ZKCP [7] and ZKCPlus [27] employ the same encryption key for encrypting different queries on the same data, which poses a risk of key reselling. As highlighted in Section I-A, the use of unique keys for each query is critical to prevent key-reselling attacks in data swarming retrieval. Conversely, in a scenario where N_Q queriers request data from N_P providers for a file containing N_D blocks, ZKDET [35] necessitates an on-chain key exchange between the querier and provider, resulting in two transactions for transferring a single data block, i.e., $2N_DN_Q$ transactions in data swarming contexts. PoUDR addresses this by substituting the key exchange with a secure and verifiable encryption approach. This change eliminates

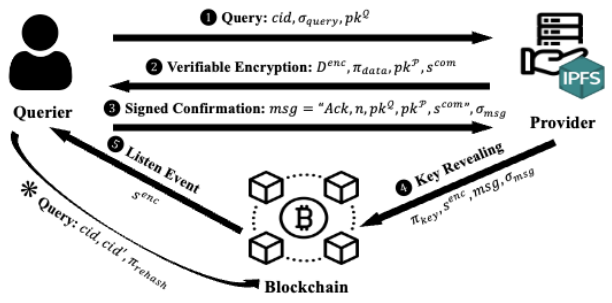
❶ Query: $cid, \sigma_{query}, pk^{\mathcal{Q}}$
❷ Verifiable Encryption: $D^{enc}, \pi_{data}, pk^{\mathcal{P}}, s^{com}$
❸ Signed Confirmation: $msg =$ "$Ack, n, pk^{\mathcal{Q}}, pk^{\mathcal{P}}, s^{com}$", σ_{msg}
Querier
Provider
IPFS
❺ Listen Event
s^{enc}
❹ Key Revealing
$\pi_{key}, s^{enc}, msg, \sigma_{msg}$
✻ Query: cid, cid', π_{rehash}
Blockchain

Fig. 2. The PoUDR protocol for a single block retrieval.

the need for on-chain key exchanges, thereby reducing the transaction count to precisely one for each data block transfer. Moreover, this design is also in line with the batched key-revealing proof enhancement on the provider side, effectively lowering the transaction number to just N_P in data swarming scenarios.

The PoUDR protocol, as illustrated in Figure 2, operates in conjunction with a blockchain system. The blockchain system is modeled as a trusted arbiter whose security is out of the scope of this paper.

The PoUDR protocol is formally defined as a tuple: $(Sig, Hash, Hash_{zk}, Commit, \mathcal{E}_{data}, \mathcal{E}_{key}, P_{enc}, V_{enc}, P_{key}, V_{key}, P_{rehash}, V_{rehash})$. Here, Sig represents a signature scheme, $Hash$ is the ZK-unfriendly hash function used in Decentralized Hash Table, $Hash_{zk}$ is a ZK-friendly hash function, and $Commit$ is a commitment scheme. $\mathcal{E}_{data}$ stands for a private key encryption scheme, while $\mathcal{E}_{key}$ denotes a public key encryption scheme. P_{enc} and P_{key} are designated as two ZK provers, focusing on encryption and key revealing proof, respectively. V_{enc} and V_{key} are the corresponding ZK verifiers for these proof systems. P_{rehash} and V_{rehash} are the recursive prover and verifier in the Rehashing Protocol. Within the PoUDR protocol, the process by which data queriers acquire information from data providers is divided into five distinct phases: Rehashing Phase, Query Phase, Verifiable Encryption Phase, Signing Confirmation Phase, and the Key Revealing Phase. The Rehashing Phase is an optional phase since this phase is only necessary for the newly uploaded data.

If a data provider intends to supply brand new data D, participation in the Rehashing Phase (Phase ✻) is required. During the Rehashing Phase, the provider must execute the Rehashing Protocol III-C, thereby registering a ZK-friendly content ID cid' for the new data on the blockchain.

In the Query phase (Phase ❶ and shown in lines 1 and 2 in Protocol 2) of the PoUDR protocol, a querier $\mathcal{Q}$ transmits a data request to the relevant providers $\mathcal{P}$. The protocol aligns with the data querying method utilized by the IPFS system. This method involves the use of a Decentralized Hash Table (DHT), which necessitates a content ID cid as the search key to identify providers holding the requested data. Under the PoUDR protocol framework, the data query must include not just the content ID but also a signature σ_{query}, authenticated by the public key associated with the on-chain account address of the data querier. This requirement is vital as it provides the data provider with knowledge about the identity and pertinent specifics of the data querier.

Following the establishment of a connection between $\mathcal{Q}$ and $\mathcal{P}$, the subsequent stage is the Verifiable Encryption Phase (Phase ❷, lines 1 to 9 in Protocol 3), during which encrypted data is transmitted. First, $\mathcal{P}$ should compute a zk-friendly hash value cid' such that $cid' = Hash_{zk}(D)$, which will be utilized in proving the correctness of encryption. Encrypting the data D by an encryption key s before transfer ensures that the data querier remains unaware of the content until payment is made. Simultaneously with the data transfer, the data provider must also validate the encrypted data, D^{enc}'s accuracy to the querier by sending a Zero-Knowledge Proof π_{data}. The proof π_{data} demonstrates the following statement: $cid' == Hash_{zk}(D) \wedge s^{com} == Commit(s) \wedge D^{enc} == \mathcal{E}_{data}.Enc(s, D)$ where cid', s^{com}, and D^{enc} are public witnesses, and s, D are secret witnesses. $\mathcal{P}$ directly proves the ZK-friendly hash because the Rehashing Protocol already ensures the alignment between cid and cid'. Upon receiving the encrypted data and its associated proof, $\mathcal{Q}$ first checks the mapping value of cid' on-chain to ensure it matches cid. She then verifies the proof's validity and then returns a signed confirmation message (Phase ❸, line 3 to 6 in Protocol 2) to the data provider. This signed confirmation message acts as an authorization of payment by the data querier, acknowledging receipt of the encrypted data. The data provider will later use this confirmation to claim payment on the blockchain.

When the data provider receives the confirmation message, the procedure advances to the Key Revealing Phase, illustrated in Phase ❹ (lines 10 to 15 in Protocol3). In this phase, it is incumbent upon the data provider to unveil the encryption key s to the querier. The data provider publishes the encryption key on the blockchain, ensuring the exclusive access to the querier, accomplished via public key encryption using the querier's public key $pk^{\mathcal{Q}}$. Concurrently, a zero-knowledge key revealing proof π_{key} is provided to authenticate the encryption's accuracy and to confirm the consistency of the prior key commitment established in Phase ❷. The proof π_{key} guarantees the truth of the following statement: $s^{enc} = \mathcal{E}_{key}.Enc(pk^{\mathcal{Q}}, s) \wedge s^{com} = Commit(s)$, where $pk^{\mathcal{Q}}$, s^{enc}, and s^{com} are public witnesses, with s being the sole secret witness. Upon the key's revelation and the subsequent verification of the proof, the smart contract autonomously processes the payment from the querier to the provider, thereby finalizing the data exchange for an individual data block.

C. Rehashing Protocol

As mentioned above, the Rehashing Protocol creates a mapping from a ZK-unfriendly hash to a ZK-friendly hash of the same data D. This protocol allows data providers to prove a ZK-friendly hash instead of an unfriendly hash in π_{data}.

The data provider $\mathcal{P}$ must first register the data D by generating the Rehashing proof π_{rehash}, a **Relaxed Groth16** proof, and sending it to the Rehashing Contract (shown in

Algorithm 1) to register D if it has not been registered previously. The Rehashing proof π_{rehash} demonstrates the statement:

$$cid' == Hash_{zk}(D) \ \wedge \ cid == Hash(D)$$

where cid, cid' are two public witnesses and D is a secret witness. Once $\mathcal{P}$ obtains the proof π_{rehash}, it will be submitted to the blockchain by sending a transaction and calling the **Registry** function in the Rehashing Contract 1. The contract will successfully store the relationship between cid and cid' if π_{rehash} is valid.

Afterward, $\mathcal{P}$ can prove the correctness of D by showing that $cid' == Hash_{zk}(D)$ instead of using the ZK-unfriendly hash. A data receiver $\mathcal{Q}$ can verify the data's correctness by performing an RPC call to the **CheckHash** function in Contract 1 and looking up the relationship between cid and cid'.

D. Relaxed-QAP & Relaxed Groth16

Since hash protocols like SHA256 have a clear block structure similar to block ciphers, applying recursive proof can significantly reduce the circuit size. Given the industry usage and the requirement for the proof to be easily verifiable in a smart contract, a recursive version of the Groth16 protocol is one of the most suitable proof systems. Additionally, NOVA [25] introduces a folding strategy for R1CS called relaxed R1CS. Based on the construction of relaxed R1CS, this paper introduces the Relaxed Groth16 proof system.

1) Relaxed-QAP: In the original R1CS, if there are n constraints, we first construct $z = (1, \mathbf{w}, \mathbf{x}) \in \mathbb{R}^m$ we have the following n equations, for $q \in \{1, \ldots, n\}$

$$\left(\sum_i z_i A_{i,q}\right) \cdot \left(\sum_i z_i B_{i,q}\right) = \sum_i z_i C_{i,q}$$

Then we can convert this representation to QAP as follows: First construct the polynomials $\{A_i\}_{i=1}^m, \{B_i\}_{i=1}^m, \{C_i\}_{i=1}^m$, and randomly pick n number $\{r_q\}_{q=1}^n$

$$A_i(r_q) = A_{i,q}, \quad B_i(r_q) = B_{i,q}, \quad C_i(r_q) = C_{i,q}$$

then we can convert the above equation to the following,

$$\left(\sum_i z_i A_i(r_q)\right) \cdot \left(\sum_i z_i B_i(r_q)\right) = \sum_i z_i C_i(r_q)$$

for $q \in \{1, ..., n\}$. For relaxed R1CS, we can set $z = (u, \mathbf{w}, \mathbf{x}) \in \mathbb{R}^m$. The relaxed R1CS has a structure similar to the original R1CS but introduces scalar u and error term E, therefore we can use the same trick to construct polynomials $\{A_i\}_{i=1}^m, \{B_i\}_{i=1}^m, \{C_i\}_{i=1}^m$. If there are two relaxed R1CS instances with same constraints, $(E_1, u_1, \mathbf{x_1})$ and $(E_2, u_2, \mathbf{x_2})$, and witness $\mathbf{w_1}$ and $\mathbf{w_2}$. Then, to prove these two statements at the same time, the new instance is $(E, u, \mathbf{x})$, and the new witness is $\mathbf{w}$, and $z = (u, \mathbf{w}, \mathbf{x})$ satisfying

$$\left(\sum_i z_i A_{i,q}\right) \cdot \left(\sum_i z_i B_{i,q}\right) = u \left(\sum_i z_i C_{i,q}\right) + E_q$$

where $u = u_1 + r \cdot u_2$, $\mathbf{w} = \mathbf{w}_1 + r \cdot \mathbf{w}_2$, $\mathbf{x} = \mathbf{x}_1 + r \cdot \mathbf{x}_2$, and $E_q = E_{1,q} + r[(\sum_i z_{1,i} A_{i,q}) \cdot (z_{2,i} B_{i,q}) + (\sum_i z_{2,i} A_{i,q}) \cdot (z_{1,i} B_{i,q}) - u_1(\sum_i z_{2,i} C_{i,q}) - u_2(\sum_i z_{1,i} C_{i,q})] + r^2 E_{2,q}$ for $q \in \{1, \ldots, n\}$. Then we can construct an $n-1$ degree polynomial e using $E_1, \ldots, E_n$. Using the same approach as in R1CS, we can construct polynomials $\{A_i\}_{i=1}^m, \{B_i\}_{i=1}^m, \{C_i\}_{i=1}^m$. Finally we got, for all r_q,

$$\left(\sum_i z_i A_i(r_q)\right) \cdot \left(\sum_i z_i B_i(r_q)\right) = u \left(\sum_i z_i C_i(r_q)\right) + e(r_q).$$

2) Relaxed Groth16: Let G_0, G_1, and G_T be three cyclic groups of prime order q. Let $g_0 \in G_0$ and $g_1 \in G_1$ be generators. In the following paper, the notation $[\cdot]_1, [\cdot]_2$ represents the point on the elliptic curve 1 and 2, respectively. A pairing is an efficiently computable function $e : G_0 \times G_1 \to G_T$ satisfying the following properties:

- Bilinear: For all $u, u' \in G_0$ and $v, v' \in G_1$, we have

$$e(u \cdot u', v) = e(u, v) \cdot e(u', v), \ e(u, v \cdot v') = e(u, v) \cdot e(u, v').$$

- Non-degenerate: $g_T := e(g_0, g_1)$ is a generator of G_T.

Then we can try to build the variant of Groth16 protocol based on the equation derived from relaxed-QAP.

$(\sigma, \tau) \leftarrow \textsc{Setup}(R)$: Pick $\alpha, \beta, \gamma, \delta, x \leftarrow \mathbb{F}$. Set $\tau = (\alpha, \beta, \gamma, \delta, x)$ and compute $\sigma = ([\sigma_1]_1, [\sigma_2]_2)$, where

$$\begin{aligned}\sigma_1 = \Bigg(&\alpha, \beta, \delta, \{x_i^n\}_{j=0}^{n-1}, \left\{\frac{\beta A_i(x) + \alpha B_i(x)}{\gamma}\right\}_{j=0}^{\ell}, \\ &\left\{\frac{C_i(x)}{\gamma}\right\}_{j=0}^{\ell}, \left\{\frac{\beta A_i(x) + \alpha B_i(x)}{\delta}\right\}_{j=\ell+1}^{m}, \\ &\left\{\frac{C_i(x)}{\delta}\right\}_{j=\ell+1}^{m}, \left\{\frac{x^i t(x)}{\delta}\right\}_{j=0}^{n-2}\Bigg), \\ \sigma_2 = \Big(&\beta, \gamma, \delta, \{x_i^n\}_{j=0}^{n-1}\Big).\end{aligned}$$

$\pi \leftarrow \textsc{Prove}(R, \sigma, a_1, \ldots, a_m, e)$. Pick r, s uniformly at random from $\mathbb{Z}_p$ and compute $\pi = ([\mathbf{K}]_1, [\mathbf{Z}]_1, [\mathbf{E}]_1, [\mathbf{Y}]_2)$, where

$$\mathbf{E} = e(x), \mathbf{K} = \alpha + \sum_{i=0}^{m} a_i A_i(x) + r\delta, \mathbf{Y} = \beta + \sum_{i=0}^{m} a_i Y_i(x) + s\delta$$

$$\begin{aligned}\mathbf{Z} = \frac{1}{\delta}\Big(&\sum_{i=\ell+1}^{m} a_i \big((\beta A_i(x) + \alpha B_i(x) + u C_i(x)\big) + h(x)t(x)\Big) \\ &+ Ks + Br - rs\delta\end{aligned}$$

$0/1 \leftarrow \textsc{Verify}(R, \sigma, a_1, \ldots, a_\ell, \pi)$. Accept the proof if and only if

$$\begin{aligned}&[\mathbf{K}]_1 \cdot [\mathbf{Y}]_2 - [\mathbf{E}]_1 \cdot [1]_2 = [\alpha]_1 \cdot [\beta]_2 + [\mathbf{Z}]_1 \cdot [\delta]_2 \\ &+ \Big[\sum_{i=0}^{\ell} a_i \left(\frac{(\beta A_i(x) + \alpha B_i(x)) + u \cdot C_i(x)}{\gamma}\right)\Big]_1 \cdot [\gamma]_2.\end{aligned}$$

$\hat{\pi} \leftarrow \textsc{Sim}(R, \tau, a_1, \ldots, a_\ell)$. Pick $\hat{K}, \hat{Y}, \hat{E}$ uniformly at random from $\mathbb{Z}_p$ and compute a simulated proof $\hat{\pi} = ([\hat{K}]_1, [\hat{Z}]_1, [\hat{E}]_1, [\hat{Y}]_2)$ with

$$\hat{Z} = \frac{\hat{K}\hat{Y} - \hat{E} - \alpha\beta - \sum_{i=0}^{\ell} a_i(\beta A_i(x) + \alpha B_i(x) + uC_i(x))}{\delta}.$$

3) Security Arguments for Relaxed Groth16: In this section, we show a formal security proof for the proposed relaxed Groth16 protocol.

Theorem 1. *The relaxed Groth16 protocol is a non-interactive zero-knowledge argument which holds completeness, zero-knowledge, and computational knowledge sound against affine prover.*

Proof. The proof of *completeness* is easy to verify. The proof of *zero-knowledge* is given by the construction of simulation algorithm SIM.
The proof idea of *knowledge soundness against affine prover* is "copy and paste" the proof of theorem 1 in Groth16. When using an affine prover strategy, the knowledge extractor can believe the proof $\pi = \Pi^* \cdot \sigma$ for some valid proof matrix $\Pi^* \in \mathbb{R}^{4\times(5+3n+2m)}$. Given the valid proof matrix Π^*, the knowledge extractor can treat the random value$(\alpha, \beta, \gamma, \delta, r, s)$ used during PROVE as indeterminates. Thanks to the domain separation property on the elliptic curve, the knowledge extractor can compute the witness from Π^* by canceling these indeterminates from the verification equation. The formal proof is the following:
Refer to the notation in Groth16, we use $\sigma = (\sigma_1, \sigma_2) \in \mathbb{R}^{5+3n+2m}$, and $\Pi = \begin{pmatrix} \Pi_1 & 0 \\ 0 & \Pi_2 \end{pmatrix} \in \mathbb{R}^{4\times(5+3n+2m)}$, where $\Pi_1 \in \mathbb{R}^{3\times(5+3n+2m)}$ and $\Pi_2 \in \mathbb{R}^{1\times(5+3n+2m)}$. When considering affine provers,

$$\begin{aligned}\mathbf{K} = {} & K_\alpha\alpha + K_\beta\beta + K_\delta\delta + K_\gamma\gamma + K(x) + K_h(x)\frac{t(x)}{\delta} \\ & + \sum_{i=0}^{l} K_i \frac{\beta A_i(x) + \alpha B_i(x)}{\gamma} + \sum_{i=0}^{l} K_i' \frac{C_i(x)}{\gamma} \\ & + \sum_{i=l+1}^{m} K_i \frac{\beta A_i(x) + \alpha B_i(x)}{\delta} + \sum_{i=l+1}^{m} K_i' \frac{C_i(x)}{\delta}\end{aligned}$$

for known field element $K_\alpha, K_\beta, K_\delta, K_\gamma, K_i, K_i'$ and known polynomial $K(x), K_h(x)$. We also can derive the similar form of $\mathbf{Y}, \mathbf{Z}, \mathbf{E}$. Now there is term $K_\alpha \cdot Y_\alpha \cdot \alpha^2$ when computing $\mathbf{K}\cdot\mathbf{Y}$, however this term with indeterminate α^2 does not show in the right-hand side of the verification equation, therefore $K_\alpha \cdot Y_\alpha \cdot \alpha^2$ implies that either $K_\alpha = 0$ or $Y_\alpha = 0$. We assume with loss of generality that $Y_\alpha = 0$. Similiarly, we can deduce that $K_\beta = 0$ by the term with indeterminate β^2. Also we have the coefficient of the product of indeterminate α and β to be $K_\alpha Y_\beta + K_\beta Y_\alpha = K_\alpha Y_\beta$. Therefore, we can assume $K_\alpha = Y_\beta = 1$. Now we have

$$\mathbf{K} = \alpha + K_\delta\delta + K_\gamma\gamma + K(x) + \dots$$

$$\mathbf{Y} = \beta + Y_\delta\delta + Y_\gamma\gamma + Y(x) + \dots$$

Then, the part term of $\frac{1}{\delta^2}$ missing in the right-hand side gives

$$\begin{aligned}&\left(\sum_{i=l+1}^{m} K_i(\beta A_i(x) + \alpha B_i(x)) + \sum_{i=l+1}^{m} K_i' C_i(x) + K_h(x)t(x)\right) \\ &\cdot \left(\sum_{i=l+1}^{m} Y_i(\beta A_i(x) + \alpha B_i(x)) + \sum_{i=l+1}^{m} Y_i' C_i(x) + Y_h(x)t(x)\right) = 0\end{aligned}$$

If we assume $\sum_{i=l+1}^{m} K_i(\beta A_i(x) + \alpha B_i(x)) + K_i'C_i(x) + K_h(x)t(x)) = 0$, then from indeterminate term

$$\alpha \cdot \left(\sum_{i=l+1}^{m} Y_i(\beta A_i(x) + \alpha B_i(x)) + Y_i'C_i(x) + Y_h(x)t(x) - E_\alpha\right) = 0 \quad (*)$$

Then for the part $\frac{1}{\gamma^2}$,

$$\begin{aligned}&\frac{1}{\gamma^2}\left(\sum_{i=0}^{l} Y_i(\beta A_i(x) + \alpha B_i(x)) + Y_i'C_i(x)\right) \\ &\cdot \left(\sum_{i=0}^{l} Y_i(\beta A_i(x) + \alpha B_i(x)) + Y_i'C_i(x)\right) = 0\end{aligned}$$

If we assume $\sum_{i=l+1}^{m} K_i(\beta A_i(x) + \alpha B_i(x)) + K_i'C_i(x) + K_h(x)t(x) = 0$, then

$$\alpha \cdot \left(\sum_{i=0}^{l} Y_i(\beta A_i(x) + \alpha B_i(x)) + Y_i'C_i(x) - E_\alpha\right) = 0 \quad (**)$$

The affine prover has negligible success probability to find nonzero E_α to satisfy both $(*), (**)$. Therefore, we can assume

$$\sum_{i=l+1}^{m} Y_i(\beta A_i(x) + \alpha B_i(x)) + Y_i'C_i(x) + Y_h(x)t(x) = E_\alpha$$

$$\sum_{i=0}^{l} Y_i(\beta A_i(x) + \alpha B_i(x)) + Y_i'C_i(x) = E_\alpha$$

Then $Y_\gamma\gamma\alpha = 0$ and $K_\gamma\gamma\beta = 0$ gives $Y_\gamma = K_\gamma = 0$. Now, we can simplify the form of K, Y to be

$$\mathbf{K} = \alpha + K(x) + K_\delta\delta, \quad \mathbf{Y} = \beta + Y(x) + Y_\delta\delta + 2E_\alpha$$

$$\begin{aligned}\mathbf{KY} - \mathbf{E} = {} & \alpha\beta + \alpha Y(x) + \alpha Y_\delta\delta + \beta K(x) + K(x)Y(x) \\ & + K(x)Y_\delta\delta + K_\delta\delta\beta + K_\delta\delta Y(x) + K_\delta + 2\alpha E_\alpha \\ & + 2E_\alpha K(x) + 2E_\alpha K_\delta\delta + Y_\delta\delta^2 - E_\alpha\alpha - E_\beta\beta \\ & - E_\delta\delta - E_\gamma\gamma - E(x) - \sum_{i=0}^{l} E_i \frac{\beta A_i(x) + \alpha B_i(x)}{\gamma} \\ & - \sum_{i=0}^{l} E_i' \frac{C_i(x)}{\gamma} - \sum_{i=l+1}^{m} E_i \frac{\beta A_i(x) + \alpha B_i(x)}{\delta} \\ & - \sum_{i=l+1}^{m} E_i' \frac{C_i(x)}{\delta} - E_h(x)\frac{t(x)}{\delta}\end{aligned}$$

The remaining part in the right-hand side of the verification equation gives us the coefficients for indeterminate $\frac{1}{\gamma}$, $\frac{1}{\delta}$ and γ are 0. This implies that $\sum_{i=0}^{l} E_i(\beta A_i(x) + \alpha B_i(x)) + E_i'C_i(x) = 0$, $\sum_{i=l+1}^{m} E_i(\beta A_i(x) + \alpha B_i(x)) + E_iC_i(x) +$

$E_h(x)t(x) = 0$, and $E_\gamma = 0$. Then we have the left-hand side to be

$$\begin{aligned}\mathbf{KY} - \mathbf{E} = &\alpha\beta + \alpha Y(x) + \alpha Y_\delta \delta + \beta K(x) + K(x)Y(x) \\ &+ K(x)Y_\delta\delta + K_\delta\delta\beta + K_\delta\delta Y(x) + K_\delta Y_\delta \delta^2 - E(x) \\ &+ \alpha E_\alpha + 2E_\alpha K(x) + 2E_\alpha K_\delta \delta\end{aligned}$$

By observing the left part in the verification equation, we can find that $E_\alpha = E_\beta = 0$ because of the domain separation of indeterminate α δ, and β. Therefore, we can simplify the above formula as

$$\begin{aligned}\mathbf{KY} - \mathbf{E} = \alpha\beta + \alpha Y(x) + \alpha Y_\delta\delta + \beta K(x) + K(x)Y(x) + \\ K(x)Y_\delta\delta + K_\delta\delta\beta + K_\delta\delta Y(x) + K_\delta Y_\delta\delta^2 - E(x)\end{aligned}$$

The term involving α now gives equation

$$\alpha Y(x) = \alpha \sum_{i=0}^{l} a_i B_i(x) + \alpha \sum_{i=l+1}^{m} Z_i B_i(x)$$

and the term involving β now gives equation

$$\beta K(x) = \beta \sum_{i=0}^{l} a_i A_i(x) + \beta \sum_{i=l+1}^{m} Z_i A_i(x)$$

If we define $a_i = Z_i$ for $i = 1, \ldots, l$, then we have

$$K(x) = \sum_{i=1}^{m} a_i \cdot A_i(x), \quad Y(x) = \sum_{i=1}^{m} a_i \cdot B_i(x)$$

The coefficient of e_i can also be extracted as the coefficients of $E(x)$. Finally, if we only observe the polynomial of x in the verification equation, we can get the original relaxed QAP. $\square$

E. Gadgets

Key-Reselling Resistance via One-time Pad over $\mathbb{F}_p$. In data swarming scenarios, a key aspect is that a single data provider might transmit multiple data blocks to one querier. Furthermore, a particular data block can be dispatched from one provider to various queriers. If the data provider does not alter the encryption key for each data request, this could lead to key-reselling attack risks. Key-reselling attacks occur when a decryption key is sold to access unpaid data blocks, meaning the querier could obtain the decryption key from a third party at a lower cost than paying the provider. As a result, the provider sends the encrypted data block but fails to receive a response or payment from the querier, who manages to decrypt the data block with the purchased key.

To counteract key-reselling attacks, employing distinct keys for encrypting each data block is essential. For this purpose, we utilize One-time Pad over finite field $\mathbb{F}_p$ (OTP-$\mathbb{F}_p$) to ensure perfect cipher security while minimizing computational expenses. OTP is an exceptionally lightweight cipher, using each encryption key only once, thereby preventing key-reselling attacks. Traditional OTP operates at the bit level, employing XOR to perfectly conceal each data block bit. However, given XOR's inefficiency in constructing constraint systems for zero-knowledge proofs, which predominantly operate on finite fields, we modify the OTP cipher to function over finite field $\mathbb{F}_p$, maintaining perfect security.

OTP-$\mathbb{F}_p$ requires converting the data block into finite field representations. Consider $\mathbb{F}_p$ with $\lfloor \log p \rfloor = L_p$. An L_D-bit data block $D \in \{0,1\}^{L_D}$ is parsed as: $d_0||d_1||d_2||...||d_L$, where $L = \lceil L_D/L_p \rceil - 1$, each $d_i \in \mathbb{F}_p$ for $i = 0, 1, ..., L$, and each d_i comprises exactly L_p bits from D to form a field element in big-endian, except for d_L. Due to big-endianness's sensitivity to trailing 0s, this parsing is distinct for each data block. After obtaining the finite field representation, OTP-$\mathbb{F}p$ generates a random key $K = k_0||k_1||k_2||...||k_L$, with each $k_i \in \mathbb{F}_p$ for $i = 0, 1, ..., L$. The cipher text $D^{enc} = d_0^{enc}||d_1^{enc}||d_2^{enc}||...||d_L^{enc}$ is created by adding each data block segment to its corresponding key segment within $\mathbb{F}_p$, i.e., $d_i^{enc} = d_i + k_i$ for $i = 0, 1, ..., L$. Note that each cipher text segment d_i^{enc} may occupy up to $L_p + 1$ bits, potentially larger than the original data block. Hence, accurately converting and parsing both the data block and cipher text is crucial.

Key Size Reduction via ZK-friendly Key Derivation Function. While OTP-$\mathbb{F}_p$ achieves perfect security with minimal field computation, its encryption key's size, potentially equivalent to the data size, could incur significant costs if disclosed on the blockchain. Generating the complete key from a small random seed via a Key Derivation Function (KDF), such as PBKDF2 [33], offers an efficient solution to shrink the key size. However, incorporating a KDF necessitates the inclusion of key generation computations in the ZK statement during the Verifiable Encryption Phase. Direct integration of PBKDF2 leads to an excessively large ZK circuit, mainly due to extensive bit-level operations like bit-wise XOR and the use of ZK-unfriendly hash functions like SHA256.

To address this issue, we adopt Poseidon [19] hash as the internal hash function for PBKDF2 and replace XOR operations in PBKDF2 with direct field addition, thus significantly reducing the ZK circuit size. We refer to this modified version as `PBKDF2-Poseidon`. The detailed structure of `PBKDF2-Poseidon` is as follows:

$$\begin{aligned}K =& \texttt{PBKDF2-Poseidon}(Seed, Salt, NumIter, Len) \quad (4)\\ =& K_0||K_1||...||K_{Len-1}, \quad (5)\end{aligned}$$

where $K_i = F(Seed, Salt, NumIter, i) = \sum_{j=0}^{NumIter-1} U_j$, and

$$\begin{aligned}U_0 =& \texttt{Poseidon}(Seed, Salt, i), \quad (6)\\ U_j =& \texttt{Poseidon}(Seed, U_{j-1}) \quad \forall 1 \le j < NumIter. \quad (7)\end{aligned}$$

Integrating `PBKDF2-Poseidon` into a ZK circuit with any constraint system type is relatively straightforward. With this alteration, the ZK statement in the Verifiable Encryption Phase modifies to:

$$\begin{aligned}&K = \texttt{PBKDF2-Poseidon}(s, Salt, NumIter, \lceil L_D/L_p \rceil) \wedge \\ &cid = Hash(D) \wedge s^{com} = Commit(s) \wedge D^{enc} = \mathcal{E}.Enc(K, D),\end{aligned}$$

where $Salt$ is an added public witness, $NumIter$ is a predefined constant, and D is the data block with L_D bits converted into $\mathbb{F}_p$ representation. Thus, only the seed s

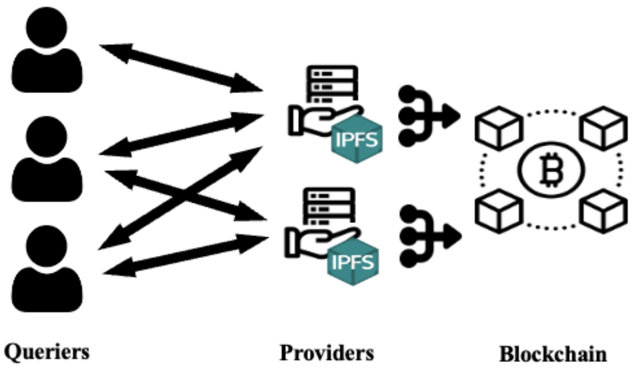

Fig. 3. Optimizing #txns via batch proof.

requires publication on the blockchain, rendering the protocol significantly more cost-effective and practical.

Optimizing #Txns via Batching. While each data block will generate only a single on-chain transaction, the volume of data blocks in practical scenarios can be substantial due to the immense size of data stored on the decentralized storage network, leading to numerous data blocks. Consequently, generating a transaction for every data block in the swarming data retrieval protocol can incur significant costs.

Given that the primary gas expense in the key-revealing transaction arises from the validation of the key-revealing proof π_{key} for an individual data block, a logical approach is to amalgamate key-revealing proofs from several blocks into a collective batch, validating them through a solitary ZK-SNARK proof. As a provider might receive requests from multiple queriers for numerous data blocks and is responsible for submitting transactions to the blockchain, grouping key-revealing proofs on a provider-centric basis becomes a sensible strategy. In particular, a provider has the option to group key-revealing proofs based on either time intervals or a predefined batch size. For instance, the provider might submit a batched proof for all blocks requested within a specific time frame T, or upon reaching a predetermined block count threshold N_{batch}, which denotes the maximum batch size, depending on practical scenarios. Consequently, the statement that needs verification in the batch key-revealing proof $\pi_{batch-key}$ transforms into:$\forall i \in \{0, 1, ..., N_{batch} - 1\}, s_i^{enc} = \mathcal{E}_{key}.Enc(pk_i^{\mathcal{Q}}, s_i) \wedge s_i^{com} = Commit(s_i)$. Utilizing a fixed maximum batch size N_{batch} is crucial to standardize the ZK circuit, eliminating the need for repeated compiling, setup, and deployment. However, the actual number of keys revealed through this batch proof might be lower than N_{batch}. This is achievable by replicating the witnesses of the last valid key as placeholder witnesses to ensure the proof's validity. Addressing these placeholder keys can be efficiently managed within smart contracts through basic duplication checks. Additionally, designing multiple circuits with varied batch sizes is a viable option to optimize efficiency and cost-effectiveness to the fullest extent. Ultimately, the provider $\mathcal{P}$ only needs to upload all valid confirmation messages with signatures and the batch key-revealing proof $\pi_{batch-key}$ to the smart contracts for verification. This method ensures that the optimal number of transactions within a given time frame T is N_P, representing the count of data providers, and is independent of the quantity of data blocks transferred. Should the transmitted blocks during time slot T exceed the maximum batch size N_{batch}, the provider can distribute these blocks across multiple batches. If the number of blocks transmitted by provider i is K_i, the total transaction count in this scenario becomes $\sum_{i=0}^{N_P-1}\lceil\frac{K_i}{N_{batch}}\rceil$. This approach effectively reduces the transaction count to $\frac{1}{N_{batch}}$ of that in a non-batching protocol. In practical applications, the number of data providers is considerably lower than the number of data blocks transmitted within the decentralized storage network, rendering the PoUDR protocol a feasible solution for current decentralized storage networks.

F. PoUDR in IPFS

Incorporating the PoUDR protocol into decentralized storage networks (DSNs), such as IPFS, can significantly boost both efficiency and fairness for data swarming retrieval. Bitswap, a core component of IPFS, manages the exchange of data blocks within DSNs. The integration of PoUDR into Bitswap implements a *Plug'n'Play* approach, allowing for seamless enhancements to the existing infrastructure of DSNs. This section will detail both the original architecture of Bitswap and the modifications for the PoUDR integration. The code structure of Bitswap, depicted in Figure 4, comprises four primary modules: Data, Message, Client, and Server.

Data. Bitswap handles data in file segment blocks, identified by content IDs, and tracks block transfers to avoid redundancies and optimize network use. It also eliminates duplicate data and unnecessary block requests, thereby boosting efficiency. This study updates data handling to better facilitate transmission beyond raw data, incorporating proofs, keys, and signatures in sent blocks and creating `getters` for accessing these elements in various code sections.

Message. Bitswap messages include want lists broadcasts, data block transfers, and block request canceling. The protocol emphasizes efficient, structured messaging to enhance network communication and ensure reliable data distribution among nodes. This study adds a `confirmMsg` field to the Message class as the receipt for encrypted data and proof, incorporates `AddEntryConfirm`, and extends the `Message_Block` subclass with proof, key, and signature fields for data block transmission and related functionalities.

Server. In Bitswap, a server is a peer responding to want lists with needed data blocks, checking local storage for availability. Peers may send requested blocks based on their exchange policies and interaction history in the Bitswap ledger, upholding network efficiency and fulfilling data requests under the protocol's fairness strategy. Server modifications entail sending encrypted data blocks, with the Engine class producing encryption proofs. The server aligns these proofs in the data block for client-side verification upon receipt.

Client. In Bitswap, the client is the peer requesting data blocks, compiling a want list of Content IDs for needed blocks

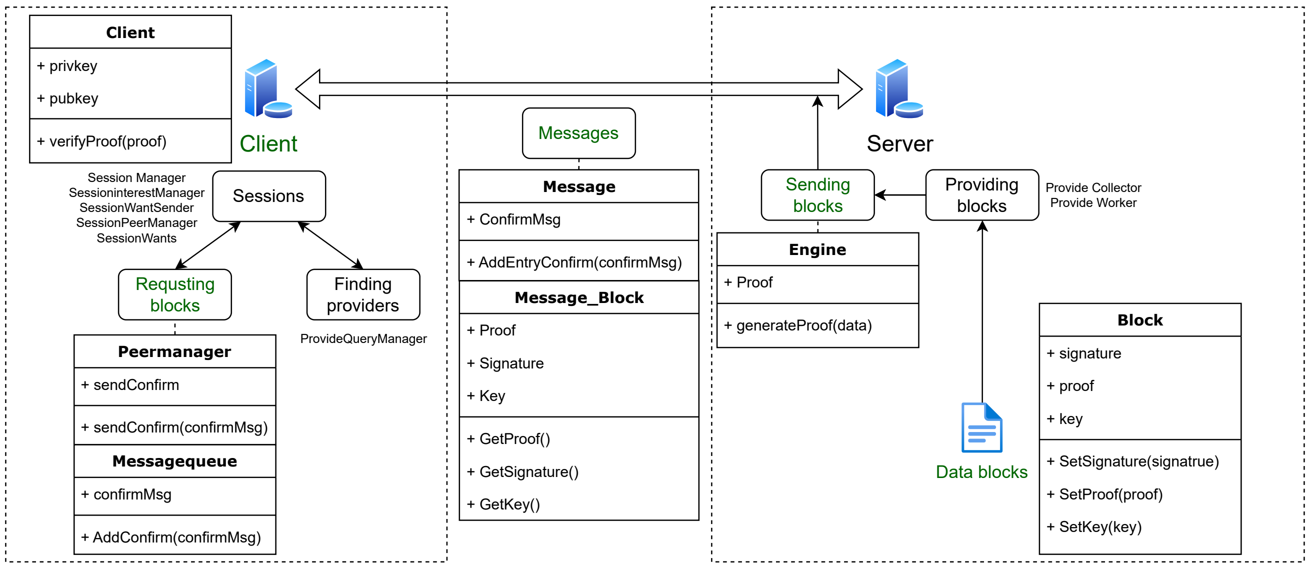

Fig. 4. Structure of the Bitswap protocol with modifications highlighted in green

and broadcasting it to network peers. Clients manage incoming blocks, duplicates, and want list updates. The Sessions class functions as the primary control on the client side, facilitating block requests and provider searches. PoUDR's client side includes private and public key fields for verifying received data and proof, with the aforementioned transmission process. Modifications also add a confirmation phase to the request mechanism for acknowledging received data.

IV. Implementation & Evaluation

A. Implementation

The PoUDR protocol was developed using Go 1.19, specifically tailored to integrate with the Bitswap protocol [22] utilized in kubo [23], a Go implementation of IPFS. The Encryption and Key-Revealing Proof circuits are constructed with the Gnark [5] framework into R1CS and further proved and verified via the Groth16 protocol [20]. Thus, the circuit size is the number of R1CS constraints in the compiled constraint system. We used Poseidon hash [19], a ZK-friendly hash function, implemented in [26] as the key commitment scheme. Additionally, we implemented the NOVA folding scheme in the Go language and modified the Groth16 algorithm in Gnark [5] to our Relaxed Groth16, compatible with Relaxed-QAP. The rehashing proof for SHA-256 is built on the Relaxed Groth16 code base. During the Key Revealing phase, providers are required to reveal the random seed s to the querier, which mandates providers to use a public key encryption scheme to encrypt s with the querier's public key $pk^{\mathcal{Q}}$. In this implementation, we chose to use Elliptic-Curve-based Elgamal encryption and developed the circuit with the elliptic curve primitives from Gnark-Crypto [8].

The Verifier Contract was implemented in Solidity and deployed on the Polygon[‡] testnet. During the period of our experiments, the price of Matic was approximately 1 USD per token, with the gas price set at 30`gwei`. All experimental evaluations were conducted EC2 instance r5a.4xlarge, which is equipped with 16 vCPUs and 128 GB of RAM.

[‡]https://polygon.technology/

TABLE II
Performance of the relaxed Groth16 ZKP protocol. $\mathbb{G}_1$, $\mathbb{G}_2$ denote a point on Curve 1 and 2 respectively. $\mathbb{E}_1$, P denote an exponentiation operation on Curve 1 and pairing operation, respectively. n, m, and l represent the number of constraints, wires, and public wires in the relaxed R1CS.

Parameter	Relaxed Groth16	Groth16
CRS size	$(2n+2m)\mathbb{G}_1$, $n\mathbb{G}_2$	$(2n+m)\mathbb{G}_1$, $n\mathbb{G}_2$
Proof size	$3\mathbb{G}_1$, $1\mathbb{G}_2$	$2\mathbb{G}_1$, $1\mathbb{G}_2$
Prover's comp.	$(4m+n-l)\mathbb{E}_1$, $m\mathbb{E}_2$	$(3m+n-l)\mathbb{E}_1$, $m\mathbb{E}_2$
Verifier's comp.	$l\mathbb{E}_1$, $4P$	$l\mathbb{E}_1$, $3P$

B. Theoretical Performance

In this section, we compare the theoretical performance of the relaxed Groth16 protocol with the original Groth16 protocol. The performance metrics considered are the Common Reference String (CRS) size, proof size, prover's complexity, and verifier's complexity. These metrics are summarized in Table II. The CRS is the information between the prover and verifier. The proof size refers to the amount of data the prover sends to the verifier, which in both protocols consists of points on elliptic curves. The prover's complexity measures the computational effort required by the prover to generate a proof, while the verifier's complexity measures the computational effort required by the verifier to check the proof.

The performance of zero-knowledge proofs is primarily determined by exponentiation and pairing operations due to their computational intensity. Exponentiation involves raising a point on an elliptic curve to a scalar, which is significantly more complex than simple arithmetic operations like addition. Pairing operations, which map pairs of points from two elliptic curves to a target group, are even more computationally expensive. Therefore, the number of these operations dominates the efficiency and practicality of the protocol.

C. PoUDR in IPFS

The cost involved in transferring IPFS data blocks via the PoUDR protocol encompasses both time and monetary components. Generating proof to verify the correctness of SHA256 computation incurs costs in terms of time and the monetary expense required for computational resources. Additionally, monetary costs arise from on-chain expenses, such as gas fees when calling a smart contract. To accurately assess the cost of querying a file, an empirical analysis was conducted. This study primarily focused on two key aspects: the duration and RAM required to generate the SHA256 hash proof, and the gas expenditure for executing on-chain transactions.

1) SHA256 Proof Generation: To demonstrate the performance of Relaxed Groth16 in proving SHA256 computation, we compare the Relaxed and original versions of Groth16 across three aspects: proof generation time, maximum memory usage, and constraint number for hashing data of different sizes. Fig. 5 illustrates the performance of both versions. The absence of records for the original Groth16 when data size exceeds 64KB is due to memory usage surpassing the maximum RAM size of the experiment machine. Although Relaxed Groth16 requires slightly more time than the original Groth16 to generate the proof, the feasibility of the original Groth16 is constrained by memory usage. Relaxed Groth16 overcomes this limitation, maintaining constant memory usage and circuit size for any file size (approximately 0.32GB memory and 161,057 constraints). In practice, the proof generation time is not critical since proving SHA256 computation is only needed in the Rehashing protocol, which runs once for each piece of newly arriving data.

2) On-Chain Cost: The constraint system size for the Key Revealing Proof in PoUDR varies with batch size. Table III shows this size grows linearly with batch size. Even for a batch of 10, proof generation takes only about $0.24s$, indicating that batching can combine many proofs into one, reducing the number of proofs for smart contract verification and improving efficiency. Batching significantly reduces gas costs for on-chain verification of key revealing proofs in PoUDR. The key revealing proof includes commitment s^{com} and ciphertext s^{enc} as public witnesses. Larger batches increase public witness size, leading to more data and computations for the Groth16 verifier, thus higher gas usage. Table III shows gas requirements for various batch sizes. Graph 6 illustrates the relationship between file size (batch size$\times 256KB$) and gas costs, showing a clear cost benefit with batching. Calculating costs based on Polygon network's gas price of 30`gwei` and Matic token value at \$1 USD, the graph indicates that transferring a 2500KB file costs around \$0.077 without batching, but only \$0.02 with batching. The graph also shows gas consumption per KB, revealing a decrease in average gas usage with batching, from 1000 `gas`/KB for smaller files to 300 `gas`/KB for larger files. This highlights the efficiency of batching in reducing costs, especially for larger file transfers.

TABLE III
Performance of the Key Revealing Proof π_{key}

Batch Size	Circuit Size	Prove Time	Gas
1	8,326	0.03s	256,833
2	16,652	0.06s	307,234
3	24,978	0.10s	357,591
4	33,304	0.12s	407,970
10	83,260	0.24s	710,244

V. Conclusion

In conclusion, this paper proposes the PoUDR protocol that enhances decentralized storage networks by ensuring fair and efficient data retrieval mechanisms. Utilizing ZK-SNARKs, PoUDR maintains data integrity and reduces blockchain transactions, improving communication and cost efficiency. Its integration into IPFS's Bitswap protocol demonstrates practical feasibility, while our proposed Relaxed Groth-16 algorithm addresses the cost and complexity of zero-knowledge proofs for feasibility. This work defines Secure Swarming Data Exchange (SSDE) and offers a comprehensive security analysis. The experimental results demonstrate that the proposed Relaxed Groth16 protocol provides a scalable solution for generating SHA256 proofs with significantly lower memory usage, and batching techniques reduce on-chain verification costs.

References

[1] W. Banasik, S. Dziembowski, and D. Malinowski, "Efficient zero-knowledge contingent payments in cryptocurrencies without scripts," in *Computer Security–ESORICS 2016: 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part II 21*. Springer, 2016, pp. 261–280.

[2] J. Benet, "IPFS - content addressed, versioned, P2P file system," *CoRR*, vol. abs/1407.3561, 2014. [Online]. Available: http://arxiv.org/abs/1407.3561

[3] S. Bhattacharjee, R. D. Gopal, J. R. Marsden, and R. Sankaranarayanan, "Digital goods and markets: Emerging issues and challenges," *ACM Transactions on Management Information Systems (TMIS)*, vol. 2, no. 2, pp. 1–14, 2011.

[4] D. Boneh and V. Shoup, *A Graduate Course in Applied Cryptography*. https://toc.cryptobook.us/, 2023. [Online]. Available: http://toc.cryptobook.us/book.pdf

[5] G. Botrel, T. Piellard, Y. E. Housni, I. Kubjas, and A. Tabaie, "Consensys/gnark: v0.9.0," Feb. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.5819104

[6] B. Bunz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *2018 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2018, pp. 315–334. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP.2018.00020

[7] M. Campanelli, R. Gennaro, S. Goldfeder, and L. Nizzardo, "Zero-knowledge contingent payments revisited: Attacks and payments for services," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 229–243.

[8] Consensys, "Consensys/gnark-crypto: Gnark-crypto provides elliptic curve and pairing-based cryptography on bn, bls12, bls24 and bw6 curves. it also provides various algorithms (algebra, crypto) of particular interest to zero knowledge proof systems." 2023. [Online]. Available: https://github.com/Consensys/gnark-crypto

[9] A. De la Rocha, D. Dias, and Y. Psaras, "Accelerating content routing with bitswap: A multi-path file transfer protocol in ipfs and filecoin," *San Francisco, CA, USA*, p. 11, 2021.

[10] S. Delgado-Segura, C. Pérez-Solà, G. Navarro-Arribas, and J. Herrera-Joancomartí, "A fair protocol for data trading based on bitcoin transactions," *Future Generation Computer Systems*, vol. 107, pp. 832–840, 2020.

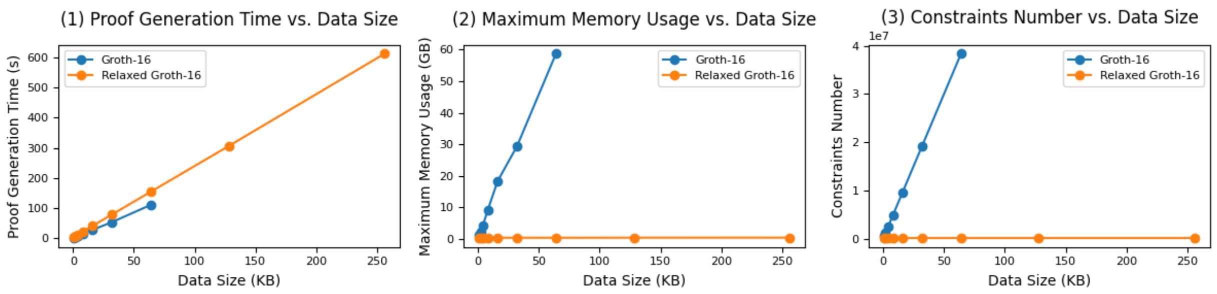

Fig. 5. Comparison between generating SHA256 proof using Groth16 and Relaxed Groth16. Graph (1) compares the two algorithms based on proof generation time for varying data sizes. Graph (2) highlights the differences in memory usage. Graph (3) shows a comparison of constraint numbers. Notably, Groth16 lacks data for sizes exceeding 64KB, as the proof generation process demands more memory than the experiment machine could provide.

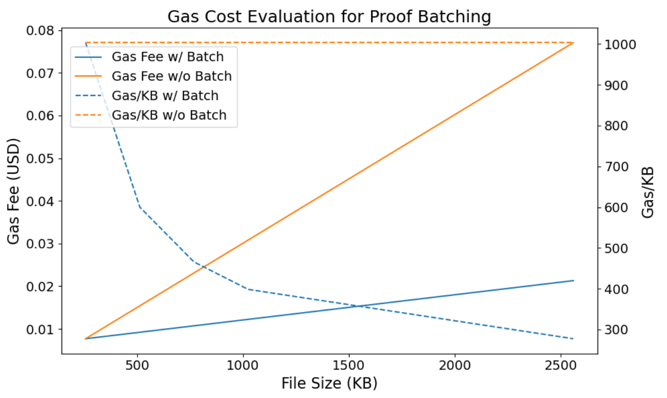

Fig. 6. Comparison of gas consumption and fees for key revealing transactions in data files of varying sizes to assess the efficiency of batching versus non-batching methods.

[11] S. Dziembowski, L. Eckey, and S. Faust, "Fairswap: How to fairly exchange digital goods," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 967–984.

[12] L. Eckey, S. Faust, and B. Schlosser, "Optiswap: Fast optimistic fair exchange," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, 2020, pp. 543–557.

[13] M. K. Franklin and M. K. Reiter, "Fair exchange with a semi-trusted third party," in *Proceedings of the 4th ACM Conference on Computer and Communications Security*, 1997, pp. 1–5.

[14] G. Fuchsbauer, "Wi is not enough: Zero-knowledge contingent (service) payments revisited," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 49–62.

[15] T. Fujitani, K. Emura, and K. Omote, "A privacy-preserving enforced bill collection system using smart contracts," in *2021 16th Asia Joint Conference on Information Security (AsiaJCIS)*. IEEE, 2021, pp. 51–60.

[16] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge," Cryptology ePrint Archive, Paper 2019/953, 2019. [Online]. Available: https://eprint.iacr.org/2019/953

[17] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, ser. STOC '85. New York, NY, USA: Association for Computing Machinery, Dec. 1985, pp. 291–304.

[18] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating computation: Interactive proofs for muggles," in *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, ser. STOC '08. New York, NY, USA: Association for Computing Machinery, May 2008, pp. 113–122.

[19] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, "Poseidon: A new hash function for {Zero-Knowledge} proof systems," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 519–535.

[20] J. Groth, "On the Size of Pairing-Based Non-interactive Arguments," in *Advances in Cryptology – EUROCRYPT 2016*, M. Fischlin and J.-S. Coron, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, vol. 9666, pp. 305–326.

[21] J. Hur, "Improving security and efficiency in attribute-based data sharing," *IEEE transactions on knowledge and data engineering*, vol. 25, no. 10, pp. 2271–2282, 2011.

[22] IPFS, "Ipfs/go-bitswap," 2023. [Online]. Available: https://github.com/ipfs/boxo/tree/main/bitswap

[23] ——, "Ipfs/kubo: An ipfs implementation in go," 2023. [Online]. Available: https://github.com/ipfs/kubo

[24] M. C. K. Khalilov and A. Levi, "A survey on anonymity and privacy in bitcoin-like digital cash systems," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2543–2585, 2018.

[25] A. Kothapalli, S. Setty, and I. Tzialla, "Nova: Recursive zero-knowledge arguments from folding schemes," in *Annual International Cryptology Conference*. Springer, 2022, pp. 359–388.

[26] Y. Li, "Liyue201/gnark-circomlib: Translate circomlib into gnark," 2023. [Online]. Available: https://github.com/liyue201/gnark-circomlib

[27] Y. Li, C. Ye, Y. Hu, I. Morpheus, Y. Guo, C. Zhang, Y. Zhang, Z. Sun, Y. Lu, and H. Wang, "Zkcplus: Optimized fair-exchange protocol supporting practical and flexible data exchange," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 3002–3021.

[28] F. Liang, W. Yu, D. An, Q. Yang, X. Fu, and W. Zhao, "A survey on big data market: Pricing, trading and protection," *Ieee Access*, vol. 6, pp. 15 132–15 154, 2018.

[29] C. Lundkvist, "An introduction to ipfs," Jul 2020. [Online]. Available: https://medium.com/@Consensys/an-introduction-to-ipfs-9bba4860abd0

[30] S. Maitra and D. J. Wu, "Traceable prfs: full collusion resistance and active security," in *IACR International Conference on Public-Key Cryptography*. Springer, 2022, pp. 439–469.

[31] G. Maxwell, "Zero knowledge contingent payment - bitcoin wiki," 2011. [Online]. Available: https://en.bitcoin.it/wiki/Zero_Knowledge_Contingent_Payment

[32] ——, "The first successful zero-knowledge contingent payment," 2016. [Online]. Available: https://bitcoincore.org/en/2016/02/26/zero-knowledge-contingent-payments-announcement

[33] K. Moriarty, B. Kaliski, and A. Rusch, "PKCS #5: Password-Based Cryptography Specification Version 2.1," Internet Engineering Task Force, Request for Comments RFC 8018, Jan. 2017.

[34] H. Pagnia, F. C. Gärtner *et al.*, "On the impossibility of fair exchange without a trusted third party," Citeseer, Tech. Rep., 1999.

[35] R. Song, S. Gao, Y. Song, and B. Xiao, ": A traceable and privacy-preserving data exchange scheme based on non-fungible token and zero-knowledge," in *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, 2022, pp. 224–234.

[36] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song, "Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation," in *Advances in Cryptology – CRYPTO 2019*, ser. Lecture Notes in

Computer Science, A. Boldyreva and D. Micciancio, Eds. Cham: Springer International Publishing, 2019, pp. 733–764.

[37] J. Zhang, T. Xie, Y. Zhang, and D. Song, "Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 859–876.

Appendix
Security Arguments

In this section, we present the detailed security proof for the critical components and the **PoUDR** protocol as outlined below.

Theorem 2 (OTP-$\mathbb{F}_p$ is a perfectly secure Shannon cipher)**.** *Let $\mathbb{F}_p$ denote the finite field over modulus p where p is a prime and $\lfloor \log p \rfloor = L_p$. Let $\mathcal{M} = \{0,1\}^M$ denote the message space, $\mathcal{K} = \mathcal{C} = \mathbb{F}_p$ denote the key space and the cipher text space, respectively. Let $\mathcal{E} = (E, D)$ denote the OTP-$\mathbb{F}_p$ cipher where the encryption E and decryption D work as follows:*

- *For any message $m \in \mathcal{M}$ and any key $k \in \mathcal{K}$, $c = E(k, m) = m + k$, where the addition works over $\mathbb{F}_p$.*
- *For any cipher text $c \in \mathcal{C}$ and the corresponding decryption key $k \in \mathcal{K}$, the decryption first calculates $m' = D(k, c) = c - k$, where the subtraction also works over $\mathbb{F}_p$. Then the decryption checks if $\lfloor \log m' \rfloor < L_p$. If true, accept m' as the decrypted message, otherwise, abort.*

The Shannon cipher $\mathcal{E}$ defined above is perfectly secure.

Proof. According to Theorem 2.1 in [4], the perfect security of $\mathcal{E}$ is equivalent to the following statement: for each $c \in \mathcal{C}$, a specific N_c exists, ensuring that for any $m \in \mathcal{M}$, $|\{k \in \mathcal{K} : E(k, m) = c\}| = N_c$. To establish this, demonstrating that $N_c = 1$ for all $c \in \mathcal{C}$ within OTP-$\mathbb{F}_p$ is sufficient. This requires computing the quantity of k satisfying $k + m = c$ for a given $c \in \mathcal{C}$ and any $m \in \mathcal{M}$. Simple field operations yield a distinct $k = c - m$, signifying a sole viable $k \in \mathcal{K}$ for each message $m \in \mathcal{M}$. Hence, $N_c = 1$ for every $c \in \mathcal{C}$, confirming the perfect security of $\mathcal{E}$. □

Theorem 3. *The PoUDR protocol delineated in Section III-B is a Secure Swarming Data Exchange (SSDE) protocol.*

Proof. It is sufficient to prove that the *PoUDR* protocol satisfies the following two security properties:

- **Data Correctness:** Considering each data block query is independent, this proof focuses on verifying the data correctness for a single data block. The proof involves two attack games, $\mathtt{Att}_{data}$ and $\mathtt{Att}_{key}$, played between an honest data querier, $\mathcal{Q}$, and a deceitful data provider, $\mathcal{P}$. The blockchain system is assumed to be a reliable arbiter, $\mathcal{J}$, and its security is not the focus of this paper.

  $\mathtt{Att}_{data}$ unfolds in the following manner:
  - Querier $\mathcal{Q}$ queries a content ID cid from provider $\mathcal{P}$, corresponding to the data block d where $Hash(d) = cid$.
  - Provider $\mathcal{P}$ prepares another data block d' such that $d' \neq d$. Then, $\mathcal{P}$ selects a seed $s \in \mathbb{F}_p$, generates k = PBKDF2-Poseidon(s), and encrypts d' to get $d^{enc'} = \mathtt{OTP}\text{-}\mathbb{F}_p(k, d')$. Then, the provider $\mathcal{P}$ generates an encryption proof π_{enc} for d' and sends both π_{enc} and the public witnesses cid, $s^{com'}$, and $d^{enc'}$ to querier $\mathcal{Q}$. Here, the provider $\mathcal{P}$ chooses to set cid instead of cid' as the public witness to pass the consistency check.
  - Querier $\mathcal{Q}$ outputs a bit b_0 to indicate whether the encryption proof is acceptable.

  The provider's advantage is quantified as: $\mathrm{DC}\mathtt{Adv}_{data} = \Pr\{b_0 = 1\}$. Let W_1 and W_2 denote the event that $Hash(d) = Hash(d')$ and $cid' \neq cid \wedge V_{enc}(\pi_{enc}, cid, s^{com'}, d^{enc'}) = 1$, respectively. The probability of $\mathcal{Q}$ accepting the proof is:

$$\Pr\{b_0 = 1\} = \Pr\{W_1 \vee W_2\} \leq \epsilon_{hash} + \epsilon_{zk-soundness}. \tag{8}$$

  $\mathtt{Att}_{key}$ is structured as follows:
  - Querier $\mathcal{Q}$ communicates the content ID cid to provider $\mathcal{P}$, corresponding to $Hash(d) = cid$.
  - The provider $\mathcal{P}$ selects two different seeds $s, s' \in \mathbb{F}_p$ such that $s \neq s'$, generates k = PBKDF2-Poseidon(s), and encrypts d to get d^{enc} = OTP-$\mathbb{F}_p(k, d)$. Then, the provider $\mathcal{P}$ generates an encryption proof π_{enc} for d and sends both π_{enc} and the public witnesses cid, s^{com}, and d^{enc} to querier $\mathcal{Q}$. The provider $\mathcal{P}$ can also send $s^{com'}$ instead of s^{com} in order to open s' later.
  - The querier $\mathcal{Q}$ verifies the proof and sends the signed confirmation message $msg||\sigma_{msg}$.
  - The provider $\mathcal{P}$ prepares and sends the key revealing proof π_{key} along with public witnesses $pk^{\mathcal{Q}}$, $s^{enc'}$, $s^{com'}$, and querier's message $msg||\sigma_{msg}$ to the arbiter $\mathcal{J}$.
  - The arbiter $\mathcal{J}$ sends the querier $\mathcal{Q}$ an encrypted seed s^{enc}. If the verification fails, $s^{enc} = \perp$. The querier $\mathcal{Q}$ does all the decryption steps to get data d. If any of the steps fails, $d = \perp$. Then, $\mathcal{Q}$ outputs a bit b_1 to state if $Hash(d) = cid$ and the verification of π_{enc} passes.

  The provider's advantage in this scenario is defined as $\mathrm{DC}\mathtt{Adv}_{key} = \Pr\{b_1 = 1\}$. Let W_3 denote the event that $s^{com} = Commit(s')$, W_4 denote the event that $s^{com} \neq s^{com'} \wedge V_{enc}(\pi_{enc}, cid, s^{com'}, c) = 1$, and W_5 denote the event $s^{com} \neq s^{com'} \wedge V_{key}(\pi_{key}, pk^{\mathcal{Q}}, s^{enc'}, s^{com'}) = 1$. We have

$$\begin{aligned}\Pr\{b_1 = 1\} &= \Pr\{W_3 \vee W_4 \vee W_5\} \leq \Pr\{W_3\} + \Pr\{W_4\} \\ &+ \Pr\{W_5\} \leq \epsilon_{commit-binding} + 2\epsilon_{zk-soundness}.\end{aligned} \tag{9}$$

  Thus, the total advantage of the adversarial provider $\mathcal{P}$ is

$$\begin{aligned}\mathrm{DC}\mathtt{Adv}_{total} &\leq \mathrm{DC}\mathtt{Adv}_{data} + \mathrm{DC}\mathtt{Adv}_{key} \\ &\leq \epsilon_{hash} + \epsilon_{commit-binding} + 3\epsilon_{zk-soundness},\end{aligned} \tag{10}$$

  which is negligible when a secure hash function, a computationally binding commitment scheme, and a secure ZK protocol are adopted. This proves that the PoUDR protocol is capable of ensuring data correctness.
- **Guaranteed Payment:** For the case of retrieving a single block, we examine two additional attack scenarios involving an honest data provider $\mathcal{P}$ and a deceitful data querier $\mathcal{Q}$. These attacks, termed $\mathtt{Att}_{forge}$ and $\mathtt{Att}_{dec}$, involve the querier attempting to access data without compensating

$\mathcal{P}$, either by forging signatures or decrypting data without legitimately acquiring the necessary key.

$\texttt{Att}_{forge}$ proceeds as follows:

- The querier $\mathcal{Q}$ picks a public key on the blockchain $pk^{\mathcal{Q}'}$ with balance greater than the block price t, and tries to forge a signature σ'_{query} over the desired content ID cid without knowing the corresponding secret key $sk^{\mathcal{Q}'}$. Then, she sends $cid||\sigma'_{query}$ to the provider $\mathcal{P}$ who has the underlying data block d such that $Hash(d) = cid$.
- The provider $\mathcal{P}$ prepares the encrypted data d^{enc} and the encryption proof π_{enc}, and sends to the querier $\mathcal{Q}$.
- The querier $\mathcal{Q}$ again tries to forge a signature σ'_{msg} over the confirmation $msg =$ "$Ack||n||pk^{\mathcal{Q}'}||pk^{\mathcal{P}}||s^{com}$", and sends them to the provider.
- The provider $\mathcal{P}$ outputs a bit b_2 to decide whether to disclose the key. If b_2 is set to 1, $\mathcal{P}$ proceeds to reveal the key to the arbiter $\mathcal{J}$. Following this, the arbiter $\mathcal{J}$ is responsible for transferring t tokens from the address linked with $pk^{\mathcal{Q}'}$ to the address linked with $pk^{\mathcal{P}}$.

Define the advantage of the querier $\mathcal{Q}$ as $\text{GP}\texttt{Adv}_{forge} = \Pr\{b_2 = 1\}$. Let W_6 and W_7 denote the event that $\sigma'_{query} = Sig.Sign(sk^{\mathcal{Q}'}, cid)$ and $\sigma'_{msg} = Sig.Sign(sk^{\mathcal{Q}'}, msg)$ respectively. We can see that

$$\Pr\{b_2 = 1\} = \Pr\{W_6 \wedge W_7\} \leq \epsilon^2_{sig-forge}. \tag{11}$$

$\texttt{Att}_{dec}$ proceeds as follows:

- The querier $\mathcal{Q}$ sends the content ID cid to the provider $\mathcal{P}$ who has the data block d such that $Hash(d) = cid$.
- The provider $\mathcal{P}$ prepares the encrypted data d^{enc} and the encryption proof π_{enc}, and sends them together with the public witness cid and s^{com} to the querier $\mathcal{Q}$.
- The querier $\mathcal{Q}$ tries to decrypt c with the information from π_{enc} and s^{com} to get d'. Then she outputs a bit b_3 to indicate if $Hash(d') = cid$.

Define the advantage of the querier $\mathcal{Q}$ as $\text{GP}\texttt{Adv}_{dec} = \Pr\{b_3 = 1\}$. Let W_8 denote the event that the querier $\mathcal{Q}$ recovers s from s^{com} and W_9 denote the event that the querier $\mathcal{Q}$ recovers s from π_{enc}. We can see that

$$\Pr\{b_3 = 1\} = \Pr\{W_8 \vee W_9\} \leq \epsilon_{commit-hiding} + \epsilon_{zk}. \tag{12}$$

Thus, the total advantage of the adversarial querier $\mathcal{Q}$ is

$$\begin{aligned}\text{GP}\texttt{Adv}_{total} &\leq \text{GP}\texttt{Adv}_{forge} + \text{GP}\texttt{Adv}_{dec} \\ &\leq \epsilon^2_{sig-forge} + \epsilon_{commit-hiding} + \epsilon_{zk},\end{aligned} \tag{13}$$

which is negligible when a secure signature scheme, a computationally hiding commitment, and a secure ZK protocol are adopted. This method effectively demonstrates that the PoUDR protocol can ensure the transfer of payment from the querier $\mathcal{Q}$ to the provider $\mathcal{P}$ for a single data block.

Given the possibility of a querier querying multiple blocks from one provider, coupled with the potential for a querier to acquire the key for a data block off-chain from other queriers who have previously accessed the same block, it becomes essential to establish that the PoUDR protocol maintains its efficacy in securing payments even under these more complex scenarios. Specifically, the protocol needs to demonstrate resistance to key-reselling, ensuring that the provider $\mathcal{P}$ receives proper compensation irrespective of the number of blocks transmitted or the involvement of multiple data queriers. Consider the attack game $\texttt{Att}_{resell}$ between two colluded querier $\mathcal{Q}_A$, $\mathcal{Q}_B$ and an honest provider $\mathcal{P}$, defined as follows:

- The querier $\mathcal{Q}_A$ queries two data blocks with content ID cid_1 and cid_2 from the provider $\mathcal{P}$ who has the corresponding data blocks d_1 and d_2 such that $Hash(d_1) = cid_1$ and $Hash(d_2) = cid_2$.
- Meanwhile, the querier $\mathcal{Q}_B$ requests the data block with content ID cid_2 from the same provider $\mathcal{P}$.
- The querier $\mathcal{Q}_A$ only proceeds the payment for data block d_1 and gets the seed s_1^A. The querier $\mathcal{Q}_B$ proceeds with the payment for d_2 and gets the seed s_2^B. Then, the querier $\mathcal{Q}_B$ sends s_2^B to the querier $\mathcal{Q}_A$.
- Since the querier $\mathcal{Q}_A$ also gets the encrypted data $d_2^{enc,A}$, the encryption proof π_{enc}^A, and the commitment $s^{com,A}$ for d_2, she tries to derive d_2 from s_1^A, s_2^B, $d_2^{enc,A}$, π_{enc}^A, and $s^{com,A}$ to get d_2'. Finally, the querier $\mathcal{Q}_A$ outputs a bit b_4 to indicate if $Hash(d_2') = cid_2$.

Define the advantage of the querier $\mathcal{Q}_A$ as $\text{GP}\texttt{Adv}_{resell} = \Pr\{b_4 = 1\}$. Let W_{10} denote the event that the querier $\mathcal{Q}_A$ recovers s_2^A from $s^{com,A}$ and π_{enc}^A, and W_{11} denote the event that the querier $\mathcal{Q}_A$ decrypts $d_2^{enc,A}$ from s_1^A and s_2^B. In PoUDR, since each time the seed is chosen randomly and independently, the querier $\mathcal{Q}_A$ cannot decrypt $d_2^{enc,A}$ unless $KDF(s_1^A) = KDF(s_2^A)$ or $KDF(s_2^B) = KDF(s_2^A)$. Thus,

$$\begin{aligned}\Pr\{b_3 = 1\} &= \Pr\{W_{10} \vee W_{11}\} \leq \text{GP}\texttt{Adv}_{dec} + 2\epsilon_{kdf} \\ &\leq \epsilon_{commit-hiding} + \epsilon_{zk} + 2\epsilon_{kdf-collision},\end{aligned} \tag{14}$$

which is negligible when a computationally hiding commitment, a secure ZK protocol, and a collision-resistent KDF are adopted. In this way, we illustrate that the PoUDR protocol is capable of ensuring payment from the querier $\mathcal{Q}$ to the provider $\mathcal{P}$ in scenarios involving data swarming.

□

Remark 1 (Security Arguments for `PBKDF2-Poseidon`)**.** *Regarding the security of the altered version of PBKDF2, while proving the collision-resistance property theoretically may be challenging, we can argue that this modification does not compromise the original security of PBKDF2 in three key aspects. Firstly, the Poseidon hash function, designed to operate directly within finite fields, is frequently utilized in the construction of ZK circuits. Its use aims to reduce circuit size while still maintaining the collision-resistance attribute, as noted in [19]. Secondly, both the seed space and key space are exceedingly vast, with $s, Salt \in \mathbb{F}_p$ and $K \in \mathbb{F}_p^{\lceil L_D/L_p \rceil}$. This expansive size renders pre-computed hashing attacks extremely impractical, especially when the prime p is approximately 254 bits in size. Thirdly, as demonstrated in Theorem 2, substituting*

XOR with addition in $\mathbb{F}_p$ preserves the perfect security characteristic of One-time Pad encryption. This implies that field addition possesses an equivalent level of fusion property as XOR, further reinforcing the security strength of the modified PBKDF2.

Remark 2 (Sybil Attack Mitigation)**.** *The PoUDR protocol is susceptible to a Sybil attack targeting data providers, where a querier requests numerous data pieces from a single provider but fails to respond after receiving the encrypted data. This results in the provider expending computation and network resources to generate the encryption proof and transmit the encrypted data, only to receive no response. To counter this, the PoUDR protocol employs a strategy of mandatory time-locked staking for data queriers. This approach involves each data provider maintaining a queue for individual queriers. Upon receiving a data query, the provider can verify two key aspects: whether the querier has staked sufficient tokens to cover the payment and whether there are any outstanding unpaid queries in the querier's queue. If either of these conditions is met, the data provider has the option to decline the retrieval service for that particular querier. This measure significantly raises the economic cost of executing a Sybil attack, as attackers would be required to stake and lock a substantial amount of tokens to successfully orchestrate the attack. Through these mechanisms, the PoUDR protocol effectively mitigates the risk of Sybil attacks against data providers, safeguarding them from unnecessary losses of computation and network resources.*